2007

A web service authorization framework using a specification language

Ellora Nayak
San Jose State University

A WEB SERVICE AUTHORIZATION FRAMEWORK USING A SPECIFICATION LANGUAGE

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

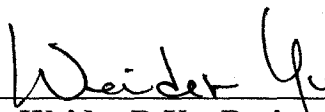
Master of Science

by

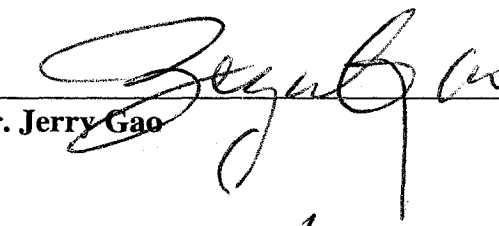
Ellora Nayak

August 2007

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING



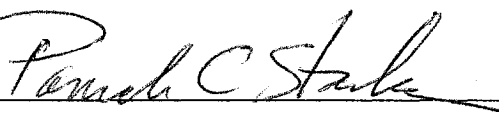
Dr. Weider D Yu, Project Advisor



Dr. Jerry Gao

Dr. Simon Shim

APPROVED FOR THE UNIVERSITY



ABSTRACT

A WEB SERVICE AUTHORIZATION FRAMEWORK USING A SPECIFICATION LANGUAGE

by Ellora Nayak

Web Services constitute an important part of distributed applications, providing flexibility in the development of distributed applications. The flexibility and platform-independence they provide in a loosely coupled architecture is truly beneficial, but also opens the door to a plethora of concerns. The need for reliable, secure, and trustworthy data access becomes more crucial. One of the key challenges in security is to determine whether an authenticated user has access to only those services for which he has authorization. It is difficult to anticipate all authorization patterns for accessing resources and, hence, difficult to define access rules beforehand. So implementing authorization becomes a concern. This paper describes an approach aimed at a more generalized and reusable solution that provides the flexibility to handle authorization rule updates in real time. The paper describes an authorization framework complemented by ARSL, a simple, flexible, robust authorization specification language based on predicate logic.

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my supervisor, Dr. Weider D Yu, Department of Computer Engineering. His understanding, encouragement, and personal guidance have provided a good basis for this thesis.

I would like to thank my husband, Manas, whose patience and encouragement enabled me to complete this work, and my parents, who have been the constant source of encouragement throughout my life.

I would also like to thank Dr. Jerry Gao and Dr. Simon Shim for being on the reading committee.

Table of Contents

List of Tables

List of Figures

1. Introduction

The internet has become a vital part of our computing infrastructure and security is a necessary part of the web's infrastructure. The use of Web Services on the World Wide Web is expanding rapidly as the need for application-to-application communication and interoperability grows. Web Services facilitate communication between various distributed software applications that deal with presenting dynamic context-based information to the user. This interoperability and flexibility poses a major threat to Web Service security, which is a major topic of research today. Enterprises are gradually getting concerned about their security; they need to secure resources accessed by their clients. For example, access to portals that provide access to a variety of resources are typical in today's enterprises, but in order to provide access to such a resource, the server must determine if the requester has the required authorization to access the resource. Authorization adds another dimension to the security issues inherent to web applications. The disadvantage of existing access control models is that their development is based on a predefined access control policy, any change to the access control policies will result in significant development effort. This implies that all security requirements (i.e., accesses to be allowed or denied) in terms of the policy enforced by the system must be specified at the beginning of system development. Specifying all protection requirements may be insignificant in smaller applications but for a larger application, specifying all requirements can become quite complex. This is because a single access control rule is not sufficient to capture all possible authorization requirements that users may need to

enforce on different data. In addition, as business requirements change the existing rules may change leading to a major overhaul of the code.

Current software development focuses on developing reusable components at all levels of application development. It is possible to quickly develop and deploy applications from readymade components by integrating them with requirement specific code. However, at levels of authentication and authorization, development of reusable components is yet to catch up. The onus is on the application developer or the deployer to implement security in the Web Services environment. This is highly error-prone. For example, a web server directs requests from all clients into a privileged application, rendering standard operating system and database access mechanisms non-functional. In such a case, threads belonging to a server process handle requests from authorized users and unauthorized users. As a result, file access authentication by the operating system and database access checks done by the database server become ineffective (Sirer and Wang 23-30). In such a case, it is the responsibility of the programmer to include code for access checks in the application to authenticate every file and database access request from every client. Another example is that security policies as well as business logic implementations change frequently and keeping them updated and synchronized is a difficult task. Due to the above reasons, there have been cases of security compromise even in highly visible and commercially stable web sites. Hence, one of the key challenges in Web Services security is designing an effective authorization model that can satisfy the unique security threats posed by a Web Service. The uniqueness lies in the fact that web based resources exposed via Web Services are typically dynamic and

distributed in nature and, hence, require adaptive authorization models that keep pace with the dynamically changing security requirements of the target enterprise.

The mechanisms required to implement access control across distributed heterogeneous domains are becoming increasingly complex. The complexity is due to the vastness of the distributed clientele accessing online resources as well as due to the variety of contextual information such as time of the day, location of the user and other factors that need to be captured at the time the request was made and incorporated in the authorization decisions. The final decision of any access control policy is greatly dependent on these context parameter values. The context is directly linked with the level of trust associated with a user and, hence, with authorizations granted to the user (Bhatti, Bertino, and Ghafoor 184-191). The access rights of users could be based on certain threshold values of the context parameters as determined by the security administrator, for example, denying access to a service after 6:00 PM. Any deviation from the threshold values would result in an automatic denial of authorization. Hence, access control policies play a vital role in ensuring Web Service security.

Access control policies determine the access of users to information on the basis of the users' identity and a collection of authorization rules which determine, for any user and any resource in the system, the types of accesses (e.g., read, write, or execute) the user is allowed on the resource (Jajodia, Samarati, and Subrahmanian 31-42). There are many different choices of access control policies: closed policy and open policy (Jajodia, Samarati, and Subrahmanian 31-42). In a closed policy, all permitted accesses are specified explicitly. When a user requests access to a service, his access rights are

verified against the specified policy for that service. If there is a policy that matches with the user's profile, access is granted else, it is denied (Jajodia, Samarati, and Subrahmanian 31-42). In an open policy, all accesses to be denied are explicitly specified. Users are allowed access to those services that they have not been explicitly denied. Some policies allow specifying both positive and negative authorizations (Jajodia, Samarati, and Subrahmanian 31-42). In such cases, conflicts may arise between authorization specifications of the same service where one rule is giving access and another rule is denying access to the same user for a given service. Conflict resolution mechanisms are required in these cases.

An access control policy defines the entities that require authorization while an access control mechanism implements the policy by ensuring that all accesses are in accordance with the underlying policy (Jajodia, Samarati, and Subrahmanian 31-42). For any access control design, separating the access control policy from access control mechanism simplifies the problem. Most access control models and mechanisms fail to maintain isolation between access control policy and access control mechanisms since their design aims at meeting the requirements of a closed policy. This tight coupling of mechanism and policy makes implementation difficult when the security requirements of an application are different from the policy. The usual solution is to implement the policy as part of the application. This solution is error prone since it makes policy modification a difficult task, requiring changes in the application code (Jajodia, Samarati, and Subrahmanian 31-42). In the proposed authorization framework, authorization rules

are the access control policy for a particular service, while the access control mechanism is based on Role Based Access Control (RBAC).

This thesis describes an authorization framework using a specification language, Authorization Rule Specification Language (ARSL) (Yu, and Nayak 54-62), as a medium to express authorization rules in order to overcome the above issues in implementing security, in particular authorization, in Web Services. The goal of the framework is to separate authorization from the application, by moving security implementation out of the domain of the developer, where it is error-prone, into the domain of security administrator. A specification language to express authorization rules complements the framework. ARSL allows users to specify the authorization policies according to which authorization decisions can be made. The proposed logical language, ARSL, based on predicate logic is simple yet sufficient to specify complex authorization rules. ARSL together with the authorization framework is flexible enough to allow real time updates to authorization rules. This approach automates the error-prone task of manually applying security checks in Web Services implementation. The framework provides separation of Web Service application implementation from the implementation of the authorization policies.

The remainder of this thesis is as follows:

- Section 2 describes the background and related work in this area.
- Section 3 lays out the design of an authorization framework using a specification language.
- Section 4 explains the Authorization Rule Specification Language.

- Section 5 describes the applicability of the framework in a real life scenario.
- Section 6 presents an algorithm to avoid conflicts in ARSL.
- Section 7 lists the main features of ARSL and its advantages.
- Section 8 compares the features of XACML and ARSL.
- Section 9 evaluates the framework in terms of scalability and performance.
- Section 10 describes the open issues and the scope of future work.
- Section 11 concludes the thesis.

The appendix contains the source code of the ARSL compiler, the generated source code, the authorization module, and the banking application, which serves as a motivation example.

2. Background

This section gives the background information leading to the thesis. The section begins with an overview of security in Web Services followed by the usage areas of Web Services where security breaches are possible and some of the security mechanisms that are used to avoid these. This section defines Web Service authorization and its role in Web Service security. It explains various Web Service authorization models and their disadvantages. This thesis is a trial to fix these disadvantages by proposing ARSL as a solution to the problem. This section ends with the list of requirements for Web Service security and a Web Service authorization model. Our solution needs to satisfy these basic requirements.

2.1. Security in Web Services

Web Services is defined as "a self-descriptive, reusable, and loosely coupled software component defined using Web Service Definition Language (WSDL), registered using Universal Description, Discovery and Integration (UDDI), and invoked using Simple Object Access Protocol (SOAP) that is exposed over the Internet" (Moorthy). Web Services has gained much popularity in enterprise application integration as a means of implementing distributed application that facilitate inter-application communication using SOAP. However, the main hindrance to this interoperability is security since the Web Service user implements security and this may vary between users.

The six core security components are (Weaver, "Security"):

- Identification – Determines the requestor.
- Authentication – Enables a system to validate the client's identity.

- Authorization –Specifies whether the client is permitted to make the request.
- Integrity – Tests if the data is reliable after the exchange.
- Confidentiality – Ensures that data is available only to authorized clients.
- Auditing – Specifies how the system logs traffic information and monitors data accesses to provide records in case of illegal access.

Web Services Usage Areas

Web Services are finding real use in areas such as application integration, and cross-enterprise integration. This wide spread usage makes security issues a major concern, requiring considerable effort from the research community.

Enterprise Application Integration (EAI) is defined as a process that involves consolidating all existing applications within an organization (Moorthy). Organizations depend on their in-house applications and other readymade IT solutions to facilitate business processes, reduce the process lifecycle, and manage resources. These applications are part of different departments and need to interact with each other to share data and services. This requires integration of the departmental applications to provide a global database of company data. At times, these departmental applications may depend on other software components to increase efficiency or to leverage readymade applications available in the market. EAI enables the development of middleware software solutions to integrate various applications within an enterprise (Moorthy).

Another area of Web Service usage is cross enterprise application integration. Integrating legacy application within an organization does not meet all of its business needs. There exists the need to integrate with clients, partners, and customer

applications. Web Services provide a means for data transfer between business partners thus ensuring that vital information is available when required (Moorthy).

Interoperability among the applications becomes an issue in using integrated solutions. In banking scenarios, banks communicate with other third party banks to maintain account clearing house (ACH) facilities. Here efficient application integration is a necessity. However, the bank cannot control or know about the infrastructure of the third party. It is not possible to decide upon the messaging formats for exchange of information and interoperability beforehand. This will result in an iterative process of redesigning to suit all partners' requirements. The cost of this type of legacy integration is very high. In such a case, Web Service helps by avoiding legacy application integration by using the Service Oriented Architecture (SOA) provided by itself (Moorthy).

Security becomes a major concern in the integrated solution described above, since they are prone to malicious attacks by professional and amateur hackers. Information available via the Internet has different levels of confidentiality depending on business requirements. Information such as company newsletters will be available to all employees while information like employee salary will only be available to the employee and his supervisor. This is where security in terms of authentication and authorization comes into play. Organizations usually secure company resources available on the network and online services by defining business roles, access rights, and other security mechanisms (Moorthy).

Existing security solutions may consist of various layers. Low-level security addresses the secure transmission of data via a network, using cryptography, and other communication security mechanisms, as described below.

Examples of Security Mechanisms

Cryptography techniques use an algorithm that takes a plain text message and a key and mathematically modifies the text so that the message is no longer in human understandable form. The original text is retrieved by using a decryption process and a key. Depending on whether a single key is used or two keys are used for encryption and decryption, there are two types of cryptography: symmetric and asymmetric (Saikali).

Symmetric algorithms use a single key for the encryption and decryption process, whereas asymmetric algorithms use a public key and a private key. Any message encrypted with the public key is decrypted only with the private key and any message encrypted with the private key can be decrypted with the public key (Saikali).

Virtual Private Network is a private communication network that allows private communications between business and individuals. They are widely used for remote access to corporate networks over a public network, such as the Internet, using standard protocols ("Network Security").

A firewall is a software program or hardware device that inspects incoming packets to a network and filters the data coming into a private network or computer. It protects the network against miscreant packets that can compromise the network's security ("Network Security").

Disadvantages of the Security Mechanisms

Web Services introduce new challenges for software application security within an enterprise by providing Enterprise Application Integration that allows application access from customer applications. Thus, network security through firewalls and packet inspection is no longer adequate. These days' firewalls can be easily penetrated and it is difficult to track illegitimate packets passing through port 80. Anyone sniffing with the appropriate tools will be able to collect confidential business information being sent as part of the messages between applications. In such a case, message-level security becomes very important for successful exchange of sensitive, business-critical information.

The application itself must address security and provide adequate mechanisms. A security model comprises of several mechanisms to enforce the desired security policy, such as:

Authentication: "establishes the identity of one party to another" (Sandhu and Samarati 241-243). Thus, authentication needs to prove the identity of a certain user to the system. Authentication may be done by a username and password.

Authorization (Access Control): determines whether a user (subject) is allowed to access an object or not. This decision is based on the system wide security policy. Authorization policies specify what a user is allowed to access.

Accounting: gathers data about system activity and data traffic. This data is then analyzed to find security breaches and infer the cause of the violations. Security administrator is responsible for regular auditing to ward off potential security loopholes

within the application or network. Accounting tracks what the user has done during his access to the system.

Due to the widespread usage of Web Services, the need to standardize methods of securing Web Services has also increased. As per the definition of Web Services, they can be dynamically discovered and used by posting XML messages over HTTP. It is a well-known fact that HTTP is vulnerable to many security attacks. Some of the problems that Web Services face can be (Moorthy):

- An unauthenticated user may try to access services that are not protected by access rules.
- An authenticated user may access services for which he is not authorized.
- Hackers might modify messages.
- The sender can non-repudiate.

2.2. Scenario to Illustrate the Authorization Problem in Web Services

A banking example described below shows the various possible client invocation scenarios and the authorization issues that can arise in accessing the banking services (Moorthy). The example demonstrates how a bank uses Web Services to provide online services to its customers through the Internet. The services provided include transferring funds between accounts, viewing account balance, online bill payment, and viewing online account statements. The clients are of four categories.

- The first category of clients has authorized access to all the banking services.
- The second category of clients can only access few services but due to lack of security mechanisms, can access other services.

- The third category of clients does not have authorized access to any of the services provided. However, they can access a service due to the lack of security mechanisms on the server side.
- The fourth category of clients include other unauthorized and unknown client who can capture the SOAP messages between any authorized client and the banking service and alter it before sending it to the server. After which he can continue the session pretending to be a legitimate user, which is a serious security breach in a banking environment.

From the above discussion, it can be seen that except the first category of clients, all others pose serious threats and can easily compromise the security of any enterprise based on Web Services. This stresses the importance of security and the need for effective security architecture in enterprise applications depending on Web Services.

Most of the currently implemented Web Service scenarios are similar to this banking application. No appropriate authorization framework exists. Organizations utilize external agencies to assess their security model. A banking scenario with authorization policies implemented using ARSL is described later sections.

Web Services security designed for a particular enterprise cannot cater to the security requirements of other enterprises. Web Services security has to be handled by a layered approach. Starting from the boundary of the enterprise network until the enterprise application level, various security mechanisms have to be analyzed. Firewall, VPN, and intrusion detection/prevention devices provide security at the network level while message-level security can be used at the application level. Organizations employ

security consultants who appraise their security requirements. Organizations dealing with security ("Verisign Web"), offer customized Web Services security assessments. They perform a comprehensive analysis of existing security architecture, and identify and prioritize security gaps as they pertain to Web Services and help to develop a cost-effective solution to close the security loopholes.

2.3. Existing Web Service Security Models

This section discusses the existing Web Service security models. The Web Services security model can be broadly classified as:

- Transport-level Security
- Message-level security

Transport-Level Security

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL) protocols provide transport-level security for the Web Services applications. TLS protocol provides server authentication and ensures a secure communication channel over the Internet between server and client. It provides security features including authentication, data integrity, and data confidentiality by using public key cryptography ("XML and Web Services Security").

Message-Level Security

To overcome the limitations of transport-level security, message-level security model can be used. There are various XML-based security standards emerging in the industry that addresses message-level security ("XML and Web Services Security"). Given below are some of the commonly used security standards:

XML Encryption

XML Encryption specification, defined by W3C, ensures data confidentiality by using encryption. The specification defines XML tags to specify encrypted data. XML encryption provides encryption of an XML file, encryption of any element, encryption of contents of any XML element, encryption of non-XML data and encryption of an already encrypted element (Siddiqui, part 2).

XML Signature

The XML Signature specification is defined by W3C and IETF. According to the W3C specification, "XML Signatures provide integrity, message authentication, and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere" (Weaver, "Enforcing" 22-24; Simon, Madsen and Adams).

WS-Security

WS-Security is the standard defined by OASIS and "enables applications to construct secure SOAP messages" (Siddiqui, part1). It provides a way to ensure data integrity, message confidentiality, and message authentication of a SOAP message. WS-Security uses XML Signature and XML Encryption specifications. It specifies methods to secure SOAP messages by the use digital signatures, message digests, and encrypted data (Siddiqui, part1).

SAML

Security Assertion Markup Language (SAML) is the standard defined by the OASIS group. It is a standard for sharing authentication and authorization information

between applications, which is the Single-Sign-On (SSO) feature. Cookies are used to implement SSO but SAML wraps the data inside XML in a standard way, thus avoiding cookies and achieving SSO interoperability ("SAML").

2.4. Access Control Models

Currently, different research groups have been trying to define access control models and policies for Web Services. Related work in the area of Web Service access control involves logical languages as described by (Bertino, Bettini, and Samarati 102-107;Coetzee and Eloff 1-11; Skalka and Wang 47-55; Indrakanti, Varadharajan, and Hitchens 774-777; Schaad, Moffet, and Jacob 3-9) trust-based context-aware models (Bhatti, Bertino, and Ghafoor 184-191). In most applications, authentication is taken care by the application itself while authorization is heavily dependent on specific access control methods such as access control lists, resulting in inflexible implementation. Adding to the above issue is the fact that the authorization requirements in networked applications are more demanding than authentication in terms of both the types of privileges required and the nature and degree of interactions between participating entities. Some of the recent work done on access control models has been motivated by the need to have a flexible and reusable solution. Other works on access control mechanisms focus on domain and type enforcement model. Languages such as WSDL (Web Services Description Language) play a significant role in describing functionality but not suitable for security specifications. Therefore, secure interoperability requires additional mechanisms.

Significant work has been done on different approaches to access control models. Web Service authorization mechanisms can be based on one of several access control models such as Role Based Access Control (RBAC), Team Based Access Control.

2.4.1.1. Role Based Access Control

RBAC (Ahn and Sandhu 207-226) uses the concept of roles for different subjects and grants access rights to subjects based on their roles. RBAC controls users' access to resources based on the access rights granted which depends on the users' role in the organization. User may or may not be granted access depending on whether he is entitled to have those privileges. In this context, a role is defined "as a set of actions and responsibilities associated with a particular working activity" (Ahn and Sandhu 207-226). Using roles simplify authorization specification; system gives authorization based on roles rather than individual user. Since roles in an organization rarely change over a period with respect to employee turnover and task reassignment, RBAC provides an effective way of minimizing the complexity, cost, and chances for error in assigning permissions to users within the organization. Roles within an organization usually have overlapping permissions. RBAC models have facilities to specify role hierarchies, where a given role can include all of the permissions of another role. Details about the authorization flow in a workflow model using mixed concepts of RBAC are discussed elsewhere (Chaari et al. 141-148).

2.4.1.2. Team Based Access Control

A team can be defined as a group of users who work together to perform tasks or achieve goals. An individual plays different roles in different teams. The Team Based

Access Control model (Alotaiby and Chen 450-454) "efficiently represents teamwork in the real world", in scenarios "where collaborative work is needed". New features are added to RBAC to support team based access control requirements. Depending on users' roles in an organization, they are allowed to join a team and are granted permissions to perform their work. Team members perform the tasks assigned to them based on their roles. Roles in the team act as a mediator to bring users and permissions to perform team tasks. Users can have different responsibilities and duties to accomplish within teams and can join teams only if they have roles that qualify them as team members. In other words, a user may not belong to a team but he should have the needed role required by a team member to perform the task. A user could belong to one or more teams depending on his roles and teams' requirements. This model has the capability to assign permissions directly to the team based on the team specification. It does not allow users to act individually but only as part of a team along with other team members (Alotaiby and Chen 450-454).

2.5. Existing Software for Access Control

Other related work in the area of Web Service access control includes eXtensible Access Control Markup Language (XACML) and .Net platform's built-in mechanism. The popularity of XML as the common mechanism for data exchange makes it suitable to find and use data from external sources. Applications can call upon automated, remotely based Web Services to provide a variety of functionalities.

2.5.1. XACML

Security Assertions Markup Language (SAML) and XACML are two derivatives of XML. SAML ("SAML") defines "how identity and access information is exchanged" and lets organizations "convey security information" to one another without having to change their own internal security architectures. However, SAML can only communicate information, whereas XACML describes how to use the information that comes in. XACML (Simon and Moses) is an OASIS standard that describes policy language and access control mechanism. XACML is an extension of XML. It has provisions for user credential verification and access control assignment based on user context. The policy language describes the access control requirements. It is possible to define new functions, data types, combining logic using XACML. XACML uses a request-response approach. The policy language is used to form a query to determine whether a given action should be allowed. The response can be one of four values: "Permit, Deny, Indeterminate (an error occurred or some required values were missing, so a decision cannot be made) or Not Applicable (the request cannot be answered by this service)" (Simon and Moses).

A major shortcoming with XACML architecture is that all access control policies must be expressed in XML, with XACML syntax. Since XACML is a verbose language, a tool is required to generate XACML to describe authorization rule. The proposed ARSL approach does not require the user to be aware of XML. Unlike XACML, that uses a request-response approach, our approach blends with existing Web Service in the form of a dynamic link library (dll). Another disadvantage is XML based data does not

fulfill the requirement of concurrent access to rules, since such data requires to be stored in an XML-database.

2.5.2. Java Security Mechanisms

Various Java libraries have provisions to address security issues, including authorization of Web Services (Narayanan and Webster). Java provides support for server applications by the Java Enterprise Edition (J2EE). This Java server platform provides support for declarative security. Declarative security means, security features are not coded in the application, rather the code related to security is defined outside the application code. At runtime, the application server interprets the security features. J2EE security does not provide a way to define access control at a lower level. Using J2EE access rules are assigned to roles at method level. The Java Authentication and Authorization Services as described in (Narayanan and Webster), provides authentication and authorization mechanisms that enables high-level security models in Java. It maps user roles to principals, while the access control evaluates the user's access rights. However, the authorization is done in the code. The code checks whether a principal is authorized to execute a code segment or not. The programmatic security aspect does not allow reusability; code has to change every time the security policies change.

2.5.3. Access Control in Microsoft's .NET Platforms

Microsoft's .NET platform has a variety of built-in security features that help in implementing security in any Web based applications ("Security Microsoft"). One of the disadvantages is .NET does not provide declarative security, which implies that security logic must be implemented in the code. The .NET framework is based on role-based

access control model, allowing access to resources based on user roles. Using this model at the application level requires code changes and involvement of application designers. This implies any modification to security policies needs modification at code level to incorporate the logic. This may be an erroneous and time-consuming procedure. Such code pieces are not readily reusable and being specific to the target application domain, the mechanisms are not appropriate enough to meet the authorization requirements of other applications.

ASP.NET Web Services has its own model of security implementation (Meier et al.). In this model, Web Service security can be applied at the following three levels:

- Platform/transport-level (point-to-point).
- Application-level.
- Message-level (end-to-end).

The concepts of platform level and message level security are as described in the previous section.

Authentication and Authorization Strategies

The ASP.NET environment together with IIS security uses the following authentication schemes to implement security in applications. It has capabilities to implement security for both Microsoft Windows and non-Windows user accounts (Meier et al.).

- Windows authentication with impersonation
- Windows authentication without impersonation
- Windows authentication using a fixed identity

Windows Authentication with Impersonation

Web Service code impersonates the IIS-authenticated caller by disabling anonymous access in IIS. Some applications require impersonation for ASP compatibility and Windows server authentication. When impersonation is allowed, ASP.NET executes with the credentials of the user or application that it is impersonating (Meier et al.).

Windows Authentication without Impersonation

With this authentication method, the application runs with the access rights of the .NET user account. Windows Access Control Lists or URL authorizations can be used for authorization requirements. URL authorization uses the *Web.config* file to specify the access rules for files. Security checks can also be coded in the Web Service application. In this case, the Web Service clients are identified via their Windows accounts, which are authenticated by the server (Schaad, Moffet, and Jacob 3-9).

Windows Authentication Using a Fixed Identity

Here the <identity> element in the *Web.config* file of the Web Service is configured to support a specific user name and password. The Web Service impersonates this fixed identity (Meier et al.).

Disadvantages of ASP.NET models

The disadvantages of the impersonation models are: difficult to implement impersonation, restricted scalability of the application, and maintainability of the access control lists (Meier et al.).

Analyzing the disadvantages of above access control models paves the way for a simpler model that can accommodate the robustness and flexibility of the existing models.

2.6. Web Service Security Requirements

Web Services require a security framework that does not hinder the exchange of data essential for their success. The following general security features broadly define the requirements of a security implementation (Mahmoud).

Authentication: Determines that the sender and receiver are who they claim to be. Username/password, smart cards, and Public Key Infrastructure facilitate authentication.

Authorization: Ensures that an authenticated client can access only those services for which they have permission. Using access control lists is one method of implementing authorization.

Availability: Authenticated and authorized users should have continuous access to their requested services even during cases of security breaches.

Confidentiality: Message confidentiality assures user that at all times data is accessible to authorized clients only.

Integrity: Ensures users that data is secure and intentional or unintentional modification is impossible.

Non-repudiation: Prevents the sender or the receiver of a message to deny sending/receiving the message.

2.7. Web Service Authorization Requirements

Given the list of Web Service security requirements in the previous section, it is required that the authorization method must not depend on any carrier protocol but it must be in accordance with Web Service protocols and lastly the authorization methodology must be generic enough to be deployed in any scenario.

The above requirements stress the importance of an authorization model that is portable across platforms and does not require application level changes to do so. A banking scenario described later shows the actual requirements of a Web Service application. In this scenario, Web Services provide platform independent communication between the business logic (banking application) at the server and the presentation tier (banking application client). Web Services provide access to sensitive data and maybe either located on one server or distributed over the Web. Client applications when used by many clients lead to a large amount of requests to the Web Services. The client application constitutes the presentation layer and is responsible only for authentication of the users while the Web Service layer handles authorization. Queries based on context parameters passed via the user request are used to fetch critical data from the Web Services. These context parameters qualify the user's request and help in determining whether the request should be granted or not. Administration of authorization data depends on the use of Web Services. The whole flow described above is generic across Web Service applications and does not use any suitable authorization framework.

2.8. Implementation of Authorization in Web Services

Currently, implementation of authorization rules in any Web Service application requires several rounds of discussion among the involved parties and stakeholders. Providing a comprehensive model of security functions and components for Web Services require the integration of currently available processes and technologies with the evolving security requirements of future applications. It demands unifying concepts. It requires solutions to both technological and business process and the coordinated efforts by platform vendors, application developers, network and infrastructure providers, and customers.

At least five parties can be assumed to involved in the decision and implementation process. They are:

- The first party is the Web Service user or requester, requesting new services.
- The second party is the organization or Web Service provider. They analyze the requested Web Service and develop the application requirements.
- The third party is a Security Architect who describes the relationships between security and the key components of Web Services architecture. He analyzes the authorization requirements and guides the Web Service provider on security implementation.
- The fourth party is an independent software vendor who describes interface and integration points between the various products in Web Services architecture.

- The fifth party validates the application. He verifies whether the application meets the requirements of the user, the business, and the security needs of the organization.

This involves collaboration among the five teams as shown in Figure 1.

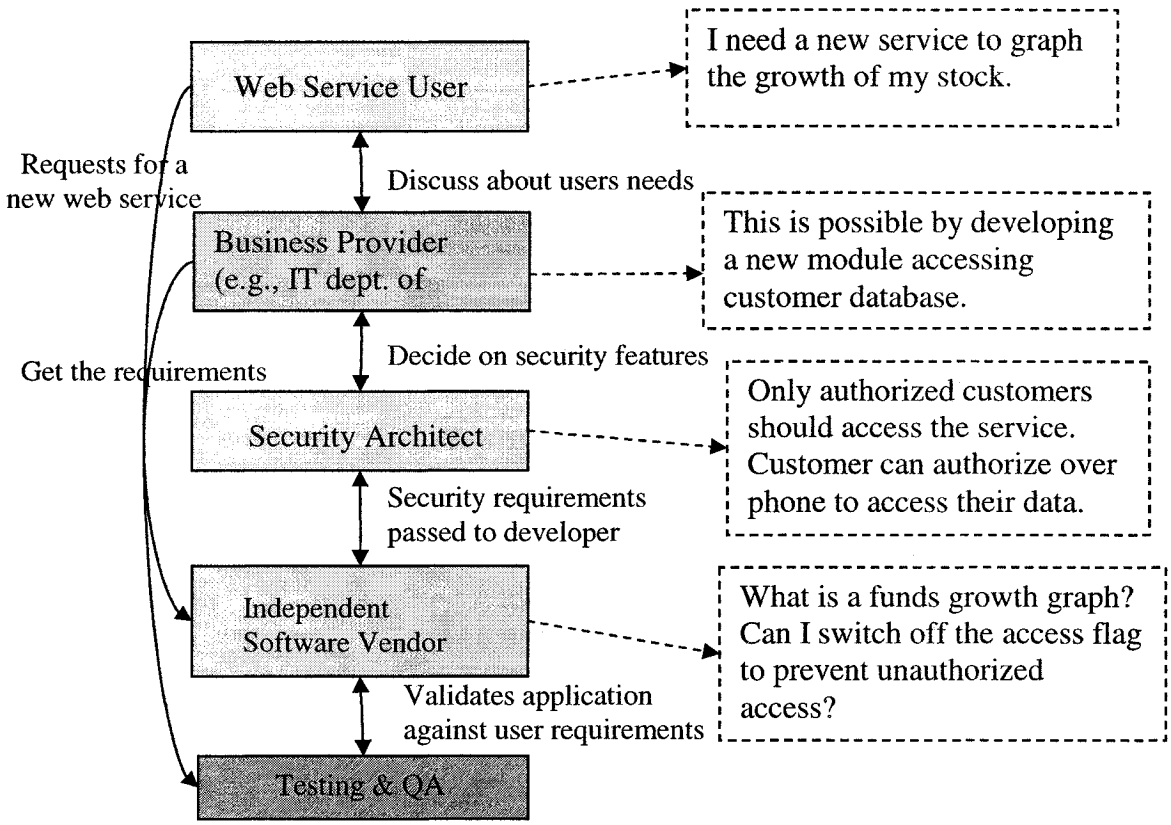


Figure 1: Current security implementation model

In such a scenario, the problems encountered can be any or all of the following:

- Members of each team might have different perspective of the requirement, different specializations, and different backgrounds.
- Often the understanding of each other's domain is weak, that results in miscommunication.
- The whole process described above involves a lot of time and effort. For example, a delay of two days in the communication flow can lead to a delay of few weeks for understanding the change required. This is highly inflexible. Given the dynamic and evolving nature of Web Services, flexibility is a necessity.

These are some of the motivating factors, to have a generalized solution for implementing authorization rules, a solution leading to separation between authorization policies and the Web Service application.

2.8.1. Design Issues in Implementing Security

Apart from the above motivating factors, other issues that need to be addressed are:

- The types of authorization information, ranging from static and generic information such as identities to specific information such as roles of the user to be used in the decision-making, need to be considered.
- The class of authorization policies that need to be supported in the Web Services architecture need to be addressed.
- Depending on the type of information, there can be different places where the information needs to be verified by authorized personnel. This authorization information can be 'pushed' or 'pulled' depending on the type of information.

- Related to the different types of information is the replication strategy and management strategy, that determines who updates and manages what information and policies. Depending on the design, different authorization architectures can be implemented which have to be suitably integrated with the Web Services architecture.

The solution to the above-mentioned problems lies in the proposed framework to implement authorization as a disjoint layer to Web Services. The rule specification language, ARSL, is part of the framework that can be used by users to express their security policies. The language comprises of different kinds of rules: *derivation rules*, which allow derivation of implicit authorizations from those explicitly specified; *access control rules*, which regulate access control policy decisions; and *integrity rules*, which are used to express different kinds of constraints on the specification and use of authorizations.

3. Authorization Framework Using a Specification Language

Similar work in this area uses an inference-based engine to parse the language used to specify authorizations (Yu and Mansukhani 316-321). The approach described in this thesis ARSL as part of a framework to solve the existing problems in implementation of authorization of Web Services. The design process involved designing of two components – a framework and a language. The framework will help in separating authorization implementation from the application development while the language will shift the onus of authorization implementation from the developer to the decision maker i.e., the security administrator.

3.1. Design of the Framework

The three basic requirements of the framework are:

1. The authorization module should be encapsulated and separated from rest of the Web Service application to avoid one affecting the other inadvertently.
2. Given a set of authorization rules, the process of generating an authorization module and integrating it with rest of the application should be automated. This should be done in a time-efficient manner so that it can be done in real-time.
3. The language used to specify the authorization rules should be simple so that the security administrator can directly express authorization rules without help of application developers.

The first requirement can be satisfied at architecture level. The encapsulation can be achieved by using the authorization module as a dynamic link library (dll). During the design phase of the application, security constraints such as authorization policies should

be well articulated. A naming standard of the authorization module APIs should be decided during the design phase. This will help in knowing which authorization functions to call for a given Web Service request. This facilitates maintainability by avoiding changes in rest of the application when authorization rules or the Web Service name changes.

The second requirement can be satisfied by proper implementation. Flex, Bison (Levin, Mason, and Brown) and gmake were chosen to generate a compiler for the authorization language, *ars.exe.* The compiler generation is a one-time effort. Whenever the authorization rules change or new rules are added to the input file, the input file is compiled using *ars.exe* to generate code in a chosen high-level language (HLL). This HLL code is compiled using a HLL compiler such as *csc.exe* to generate a dll for authorization.

The third requirement can be fulfilled by developing a language. This language is in Backus-Naur Form (BNF) that can be generated by the chosen compiler generator, Bison.

The details of the language design are given in the following section. The proposed model for specifying authorizations rules for Web Services has three major components:

- an authorization policy
- a compiler
- generated code

The block diagram given below shows the relationship between the various components present in the proposed framework. First, the security administrator expresses a set of authorization rules for a Web Service using ARSL. The compiler takes the authorization rules as input and generates the authorization module. The authorization module is compiled to generate the authorization dll (AuthLib.dll). This library is dynamically linked with rest of the Web Service module at runtime.

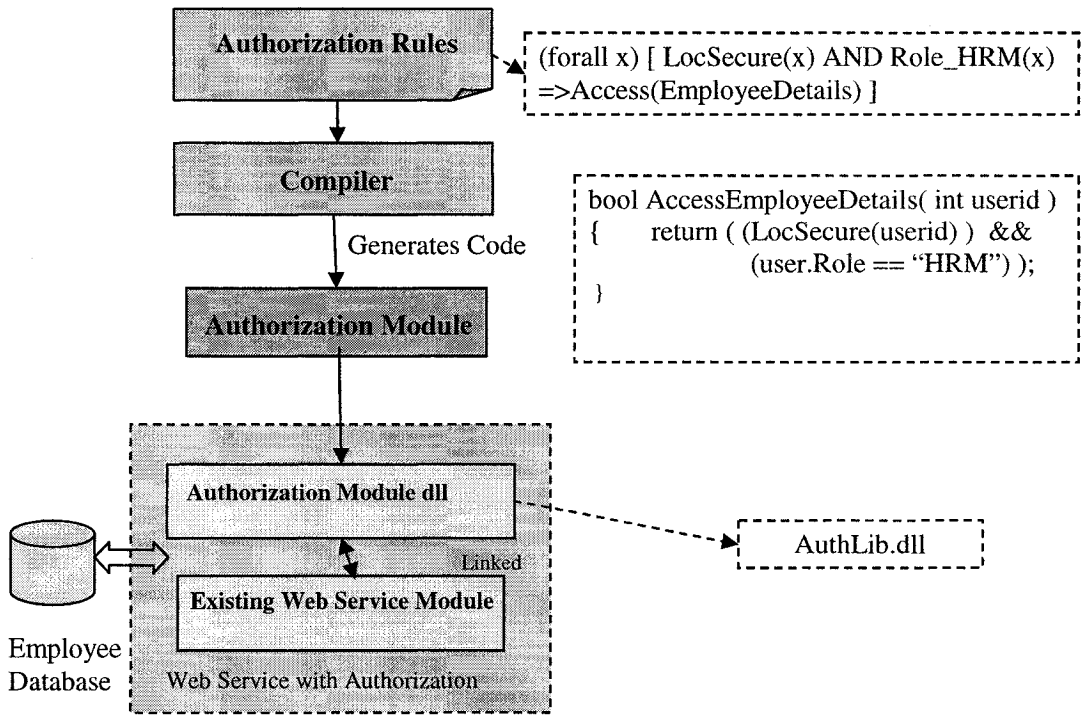


Figure 2: Block diagram showing the relationship between various components of the framework

Figure 3 shows how the information flows from the time a user requests access to a Web Service till the time the request is processed by the Web Service module after proper authentication and authorization of the user.

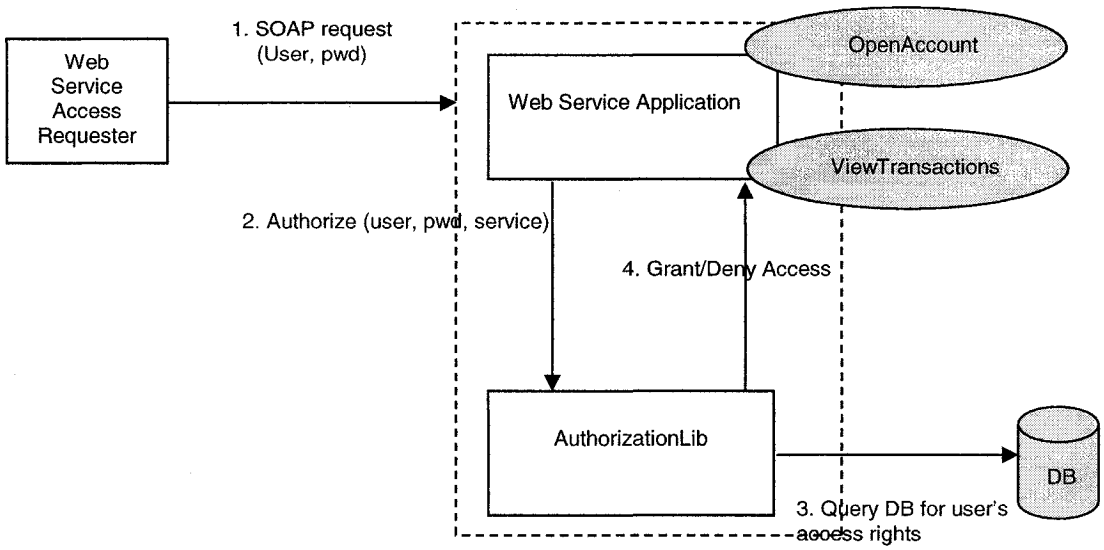


Figure 3: Data flow diagram of the authorization framework

3.2. Design of ARSL

The Authorization Rule Specification Language complements the proposed authorization framework. This language is based on predicate logic. Predicate logic is a "well-understood formal language, with well-defined syntax, semantics, and rules of inference" (Cawsey). As mentioned before, predicate logic is an important knowledge representation language. It is suitable for representing complex facts about the world, and deriving new facts in a way that guarantees that, if the initial facts were true then so are the conclusions.

For example, a rule that allows only employees in the sales team to view sales documents, can be represented as

(forall x) Sales_Team(x) => Access (ViewSalesDoc)

Predicate logic makes specifying general statements easy, as shown in the above example.

In general, predicates (Cawsey) are functions that are applied on things to yield atomic sentences. The truth-value of these sentences can be determined by knowing only the identity of the things to which the predicate is applied.

Quantifiers are used in sentences to indicate how any variable in the sentence should be handled. The two quantifiers in predicate logic are ***forall*** ($\forall$) and ***exists***($\exists$). Examples of quantifier usage are as follows:

$\exists$ *bird(X) => ~flies(X)*

i.e., there exists some bird that does not fly.

$\forall$ X(person(X) => $\exists$ Y loves(X,Y))

i.e., every person has something that they love.

The meaning of $\forall$ and $\exists$ is defined again in terms of the set of objects in the domain. $\forall$ X S means that for every object X in the domain, S is true. $\exists$ X S means that for some object X in the domain, S is true.

As described in (Cawsey), "the semantics of predicate logic is defined in terms of the truth-values of sentences". The truth-value of any statement in predicate logic can be determined if the truth-value of the constituent components of that statement is known. An interpretation function defines the meanings of the basic components, given some domain of objects related to the scenario.

To overcome the issues mentioned at the beginning of Section 3, our approach uses ARSL as part of a Web Service authorization framework. The goal is to develop a simplified language that can be easily used to specify authorizations. A security administrator need not be proficient in a programming language. To illustrate the

framework we consider a role based access control model. In this model, access is given based on a user's role. When a user makes a request for a service, his role is validated against his request, access is given only if his role has access to the requested service. This can be extended to other access control models too.

3.3. Requirements of an Authorization Language

The design of ARSL involved enumerating the features that a specification language should have. The language should:

- Provide a way to specify string, Boolean, numeric constants, and variables.
- Provide a way to define individual components that combine to give rules. Facilitate combining individual clauses (or macros) to derive rules for access request.
- Provide logical and arithmetic operators.
- Provide an abstraction layer that separates policy maker from the details of the Web Service application.
- Be complete enough to express any rule. Proof of completeness is given below.

Proof of Completeness

Any authorization rule can be simplified to its constituent rules combined by operators. Repeated application of simplification for each such constituent rule will finally give an expression involving atomic values (any constituent which cannot be further simplified is known as an atomic value) and operators (numerical / string / Boolean). The atomic components of the rule can be either constants or variables. As constants, and expressions involving those constants, are already supported by ARSL, our

proof of completeness rests on proving that the variables, if any, need to be database variables or system variables such as time and date. In case variables are not pre-stored values, these can change, implying that authorization rules can change without any action by security administrator. This non-deterministic authorization will fail the very purpose of implementing authorization in Web Services. Hence, by proof of contradiction these variables need to be pre-stored. The remaining part of the proof, that any such stored variable is a database entry, is evident.

In business applications stored data needs to meet some minimum criteria such as easy to organize, reliable, easy to distribute, easy to synchronize, data integrity, faster access and ability to handle large volumes of data which are the very basic requirements fulfilled by database. Therefore, any stored variable in a business application is a database entry. As Web Service applications are business applications, it proves the completeness of the proposed ARSL. Proof of completeness ensures that ARSL can be used to express any authorization rule.

3.4. Grammar Rules

Following is the simplified grammar of ARSL in Backus-Naur form.

<start>→<macro_stmt> <rules>

<macro_stmt>→

| <macro_stmt> <macro> IMPLIES VAR (VAR);

<macro> → VAR(VAR) EQUALS VAR

| VAR (VAR) NE VAR

| VAR (VAR) GE NUMBER

| VAR (VAR) EQUALS NUMBER

| VAR (VAR) NE NUMBER

| VAR (VAR) GT NUMBER

| VAR (VAR) LE NUMBER

| VAR (VAR) LT NUMBER

| NOT <macro>

| VAR (VAR)

| <macro> OR <macro>

| <macro> AND <macro>

| <macro> GE <macro>

| (<macro>)

<rules> → BEGIN <prog> END;

<prog>→

| <prog> <access_rule>

| <prog> <error>

<access_rule>→

| <access_rule> <asgn>

| <access_rule> <quantifier> [<predExpr> IMPLIES <accessExpr>];

<quantifier>→ FORALL EXISTS

| EXISTS FORALL

<asgn>→

| VAR ASSIGN VAR

| VAR ASSIGN <expr>

| <expr> ASSIGN ""VAR""

<accessExpr>→

| <accessExpr> OR <accessExpr>

| <accessExpr> AND <accessExpr>

| ACCESS (VAR)

| DENY (VAR)

<predExpr>→

| TRUE

| FALSE

| VAR

| ACCESS (VAR)

| DENY (VAR)

| (<predExpr>)

| NOT <predExpr>

| <predExpr> OR <predExpr>

| <predExpr> AND <predExpr>

| <predExpr> EQUALS VAR

| <predExpr> EQUALS ""VAR""

| VAR (VAR)

| <predExpr> GE <predExpr>

| <predExpr> GT <predExpr>

| <predExpr> LT <predExpr>

| <predExpr> LE <predExpr>

<expr>→

| VAR

| (<expr>)

| NOT <expr>

| <expr>OR VAR

| <expr> AND VAR

| <expr>EQUALS VAR

| <expr>'+' VAR

| VAR (VAR)

Figure 4 shows the stages of compiler and the input and output at each stage as relevant to ARSL.

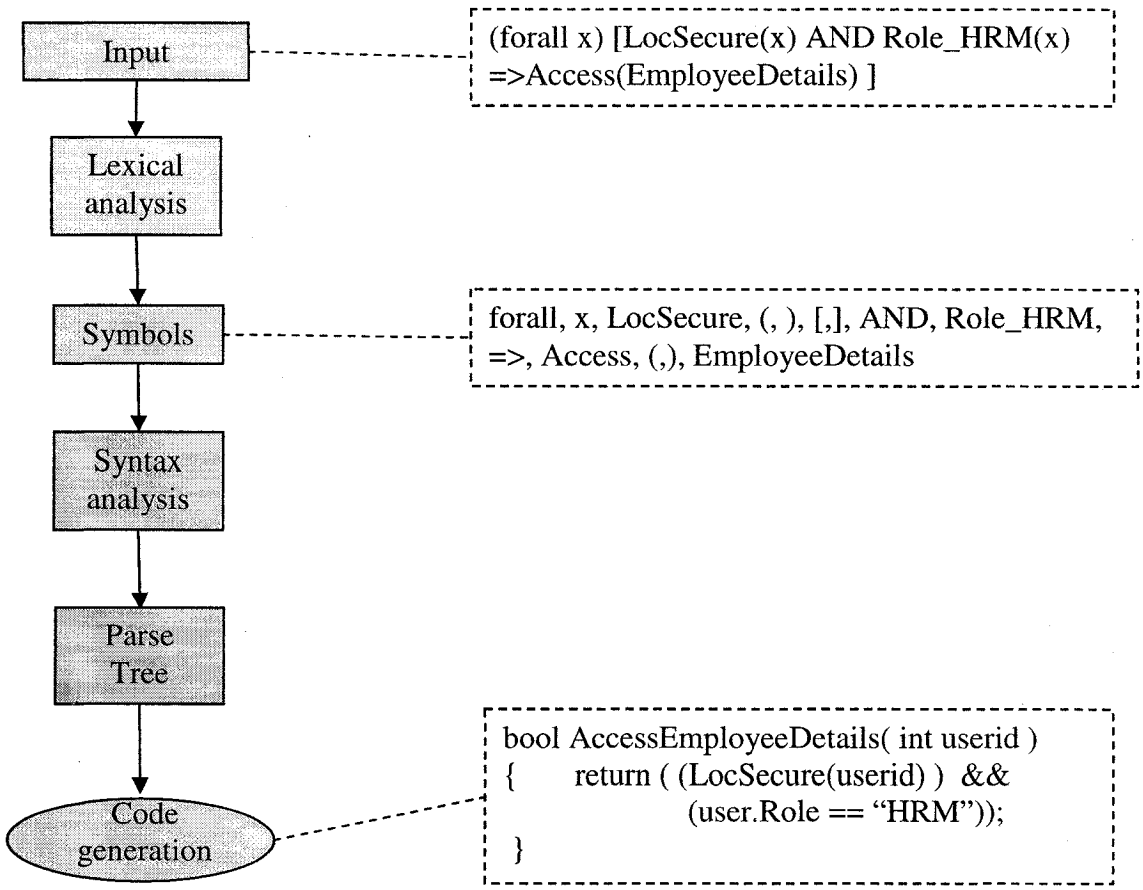


Figure 4: Different phases of the compiler with output

A typical input file consists of zero or more macros followed by a set of access rules. Each macro defines a constituent function that maybe used in the access rules that follow it.

The compiler parses the input rules to generate functions in a high-level language that implements the logic present in the rules. In general, the function generated consists of logical combination of the constituent functions present in any authorization rule for a given service. The authorization module is compiled to generate a dll. The dll is linked with the existing Web Service as an authorization layer without any changes to rest of the application. The authorization module serves as a separate layer on top of the existing Web Service thus providing reusability. With change in authorization rules or security policy of a Web Service, the authorization module can be independently changed without changing any part of the actual Web Service implementation.

When a user requests for a Web Service, the authorization module ensures that the user exists in the user database of the main Web Service application otherwise, a system error occurs and returns an appropriate error message. This assumption is allowed, because during authentication the data entered by the user will be compared to stored data about the user, if the user does not exist, authentication does not succeed. If authentication succeeds, the user's data is then fetched to determine his access rights.

4. Authorization Rule Specification Language (ARSL)

As mentioned before, ARSL is based on predicate logic. This facilitates expressing complex rules. Without using predicate logic, a user would have to write complex English sentences. For example, if user x is logging in after 9 AM and before 5 PM, and has role of a developer and is at location y then allow him access to view the project report.

Another alternative to writing complex English sentences is programming language. Expressing the above rule in a programming language would result in a deeply nested if-then-else conditional statement. If the access rule were dependent on many other functions or clauses, the nested conditional statement will get complicated. Any modification to the rules in future would be an error prone task. Predicate logic combines the simplicity of expressing in English like language with the use of logical symbols for representation.

In this section, the basic constructs of the authorization language are introduced. In subsequent sections, the usage of the language is described using an example. Some of the terminologies used are defined below.

4.1. Definitions

4.1.1. Authorization Rule

An authorization rule is of the form:

Quantifiers [(predicate1) op (predicate2) ...op (predicate3) => Access(service)];

Example:

(forall x)[(Course_Cmpe207(x) OR Course_Cmpe200(x)) AND
Spring2006_Semester(x) => Access (SOLARIS_LAB)];

This rule states that if user x is enrolled in cmpe207 course or if user x is enrolled in cmpe200 in spring 2006 semester then user x can access the Solaris laboratory. The terms on the left hand side of "=>" are used to specify conditions that must be verified for the authorization to hold true i.e., the truth-value of the right hand term depends on the truth-value of all the left hand side terms. Authorization rules are specified by the Security Administrator to specify the access policies of the hosted Web Services to requestors.

4.1.2. Derivation Rule

Derivation rules allow the derivation of authorizations based on other authorization rules. The new rules maybe derived from pre-defined rules or explicitly specified by the Security Administrator.

(forall x)[Access(WIFI) AND CmpE_Student(x)
=> Access(eJournals)];

In this example, Access (eJournals) is derived from Access (WIFI) rule, which is assumed to be defined prior to Access (eJournals). Due to their generality, derivation rules can express different kinds of implication relationships between authorizations. In the above example, the derivation of an authorization rule Access(eJouranl) is based on the presence or the absence of another authorization (Access(WIFI). By using variables and by combining different literals, we can also express a large variety of implication relationships.

(forall x) [NOT Read_Solution(x) => Access (WriteExam)];

This rule derives the negative authorization for a user to write an exam if the user has read an object of type *Solutions*. For the above input the code generated will be

```
bool AccessWriteExam( user)
{
        return ( !Read_Solution(user)) ;
}
```

4.1.3. Authorization Specification

An authorization specification can be defined as a collection of rules whose evaluation determines whether the requested access is to be allowed or refused. Accuracy of an authorization specification is very important. An authorization specification is accurate if it is both consistent and complete. Consistency means that for each access request a unique decision can be made whether to grant access or deny access but not both. Completeness implies that for each request a decision to either allow or deny exists (Jajodia, Samarati, and Subrahmanian 31-42) i.e., there is no conflict among rules. ARSL uses a conflict resolution mechanism, described later, to achieve this goal.

Consistency of rules requires that the access decision be unique. Similarly, completeness also requires unique decision to be generated. Completeness of access decision does not imply authorization specification completeness rather it ascertains the presence of a decision. It is true for default decisions where no authorization is specified or derived (Jajodia, Samarati, and Subrahmanian 31-42).

4.2. Language Constructs

The various constructs of the language are as follows:

Constants: Boolean constants (true and false), string constants (set of characters within quotes), numeric constants.

Example:

(forall x) [CurrentTime(x) > 900 && CurrentTime (x) < 1700
=> Access (BankServices)];

This authorization rule grants all employees access to the banking Web Service if the current time is between 9:00 AM to 5:00 PM, which are numeric constants.

Predicate Symbol: ARSL uses predicate symbols. A unary predicate *Access* or *Deny*, which takes as argument the service name for which the rule is created.

Access (ServiceName).
Deny (ServiceName).

During code generation, calls to the Access predicate i.e., calls to Access (ServiceName) is translated to '*AccessServiceName (UserId)*'. For an *Access* predicate used on the right hand side of "=>", code is generated for a new function with the same name. If it is on the left hand side, code generated is a call to the function. This standard naming convention makes it easier to call any access function from within the Web Service code.

Operators: Arithmetic operators, logical operators, assignment operators, string operators are implemented. Example: AND, OR, NOT, =, ==, +, - , *, >, <, >=, <=. These operators are used to join predicates, in conditional statements and in macros to define new access functions. Typically, conditional statements are applied when we need to define constraints such as, the time of day or the days of the week when a user or group can access the resource and perform a task. Conditions may contain one or more comparative statements joined using a Boolean operator for example, AND, OR. If the result of the entire expression evaluates to true, then the authorization statement is true for the selected privileges, resources, and policy subjects.

Example:

(forall x) [IdleSessionDuration(x) > 10 => Log_off(x)];

The rule states that if user has been idle for more than 10 minutes, log off the user.

Quantifiers: forall, exists

Delimiter: Semicolon terminates rules.

Macros: It is like a subroutine of an access rule that can be reused. It is defined in terms of user data stored in the user database, such as his location, role etc. These are evaluated in isolation and do not specify an access rule. A macro consists of left-hand side terms and a right-hand side term separated by the operator '=>'. The left hand side components constitute the definition of the macro given on the right-hand side. The truth-value of the macro on the right hand side depends on the components used on the left hand side. A macro constitutes the defining terms of any authorization rule. The grammar for macros is as follows.

<macro> → *<macro_stmt> IMPLIES VAR(VAR)*
<macro_stmt> →*VAR(VAR) EQUALS VAR*
| *VAR(VAR)*
| *<macro_stmt >AND< macro_stmt>*
| *<macro_stmt >OR < macro_stmt>*

A macro is composed of one or more *<macro_stmt>*. A *<macro_stmt>* maybe composed of data variables such as location that specify the constraints to be used in describing an authorization rule.

Example:

[Location(x) =="Sunnyvale" OR Location(x) =="San Jose" => LocationSecure(x)];

Where, *LocationSecure* is a macro. *Location (x)* represents a variable name or a column name 'Location' in the user table in the database. The generated code takes care of calling appropriate functions to fetch the user's data from database. The constraint described here is, if user x is located at Sunnyvale or San Jose then user x is at a secure location.

Access Rule: Access rule specifies the rules to access the resource. It can either grant access or deny access to a resource. It consists of a set of macros or defined authorization rules combined using operators provided in the language. This determines the access policy for a particular Web Service, whether to give access to a Web Service or not. Each rule consists of a left-hand side component and a right-hand side component. The left-hand side is a list of macros or other access functions, and the right-hand side is a list of services; the operator '=>' is used to separate them. Access rule includes both *Access* and *Deny* rules.

The grammar for AccessRule is as follows:

<access_rule> → *<quantifier>[<predicate> IMPLIES <access_expr>];*

<predicate> comprises of macros, prior defined access rules, and Boolean constants joined using logical operators.

<access_expr> → *ACCESS(< VAR>)*
| *DENY(< VAR>)*
| *<access_expr> AND < access_expr>*

<access_expr> gives a list of 'Access (ServiceName)'or 'Deny(ServiceName)' constructs separated by the logical operator 'AND'.

For example, consider a rule, for all students, who are enrolled in CmpE207 or enrolled in CmpE200, in the current semester have access to the Solaris laboratory. Using ARSL, this can be expressed as:

(forall x) [(Course_CmpE207(x) OR Course_CmpE200(x)) AND Current_Semester(x) => Access (SOLARIS_LAB)];

Where *(forall x)*, is the quantifier which means '*for all users X*' in this context and *Course_CmpE207(x)* and *Course_CmpE200(x)* are macros.

The code generated for the above rule is:

```
bool AccessSolarisLab( int userid )
{
        return ( Course_CmpE207(userid) && Course_Cmpe200(userid) );
}
```

Where Course_CmpE207 and Course_CmpE200 are macros or other access functions, that return a Boolean value depending on whether a user is enrolled in the course or not.

Rules can be simple, for example,

(forall x) [true => Access(INTERNET_LAB)];

Using the Boolean constant, "true", we can specify a rule to allow all users to access Internet laboratory. In general, since the semantics of predicate logic is defined in terms of the truth-values of components, the code generated from the above rules simplifies into calls to the functions defined by macros joined by the logical operators separating the rule definition.

4.3. Context Information Handling

As described earlier, in role-based access control (RBAC) model, a typical authorization policy is represented as "User x in role R has permission P". To make the

authorization framework aware of context information, context-related constraints in authorization policies can be specified. Access policies such as "User U with role R satisfying constraint C has permission P" are allowed. Here, a constraint is defined as an access restriction that can be specified by the authorization rules. Access to service S is granted to role R if and only if constraint C is satisfied.

Numerous types of contexts are possible, however, in the proposed framework we deal only with the context of the current authorization request (e.g., the status of the user making a request; the status of the object being requested; when and where the request originated). Context-based constraints help in dynamic determination of access results based upon the current context of the request, rather than based on just the role of the user.

A constraint imposes restrictions on an authorization policy. The restrictions act on some context type, the value of which determines whether a particular condition is satisfied. The context condition is expressed as a Boolean expression (e.g., time > 09:30, IPAddress = 128.67.218.13) (Weaver, "Security"). ARSL provides macros as means of expressing constraints. Macros complement authorization rules in defining access policies.

A macro has two components: the context type and context implementation. A context type is a parameter or environment variable describing the situation under which the access request is issued. Context type includes common information such as, time or location. The context implementation of a context type is a function that can evaluate the value of the context type (Weaver, "Security"). In our model, context data is handled in

the static part of the authorization module. For example, context type time is implemented as a static function that uses the system's local time function, or it could be a Web Service. The Web Service determines whether the request is valid or not by calling the corresponding *Access* functions in the Authorization library, *AuthLib.dll*. The *Access* function is internally composed of function calls to evaluate the context conditions. Each macro specifying a context condition is evaluated via its implementation and the value returned to the *Access* function. The Boolean results of all context conditions are evaluated to determine whether access is given. If the access policy is satisfied, the request is granted, otherwise it is denied.

5. A Motivating Example

A banking example is described below to illustrate the authorization framework and the proposed ARSL for specifying authorization rules. A hypothetical bank, iBank of California, uses Web Services to provide access to its banking services to its employees. The banking application is based on the above authorization framework, where the authorization module sits as a separate layer. The banking application is a .NET Web application and uses ASP.NET Web Forms and C# classes. The application uses five Web Services. These are *OpenAccount, ModifyTransaction, AfterHourAccess, Deposit, and EditAcctDetails*. The client application is the actual front-end of the banking application that uses these Web Services. The access control policies for the Web Services are written in ARSL, in a separate input file. A script is used to compile the input file and generate the code for the Authorization module, to compile the Authorization module and generate the dll and finally to link the dll with the Web Service application. When a service is requested through the interface provided via the client, the client communicates with the Web Service application to fulfill the request. The Web Service application in turn authenticates the requestor before checking for his access rights. On successful authentication, the Web Service calls appropriate functions from the authorization module to verify the access rights of the requestor. If authentication fails for any reason, access is denied to the application. The system architecture diagram for accessing *openAccount* Web Service is shown in Figure 5. Here we consider role based access control to design the authorization constraints. The simple scenario considers several roles depending on the functionality of the bank employees and the

customers of the bank. The rules specify the access rights of the users based on their roles, for the various services provided by the bank.

Figure 5 shows an employee, Bob, logging into the banking client application to start his daily job functionalities. The client application sends Bob's username and password with his request to access *OpenAcount* to the Web Service. The Web Service uses the authorization module for authentication and authorization. On successful authentication, Bob's data is fetched from the database and stored for the entire session. This data will be used for the current authorization of *OpenAccount* Web Service as well as for future authorization requests in that session.

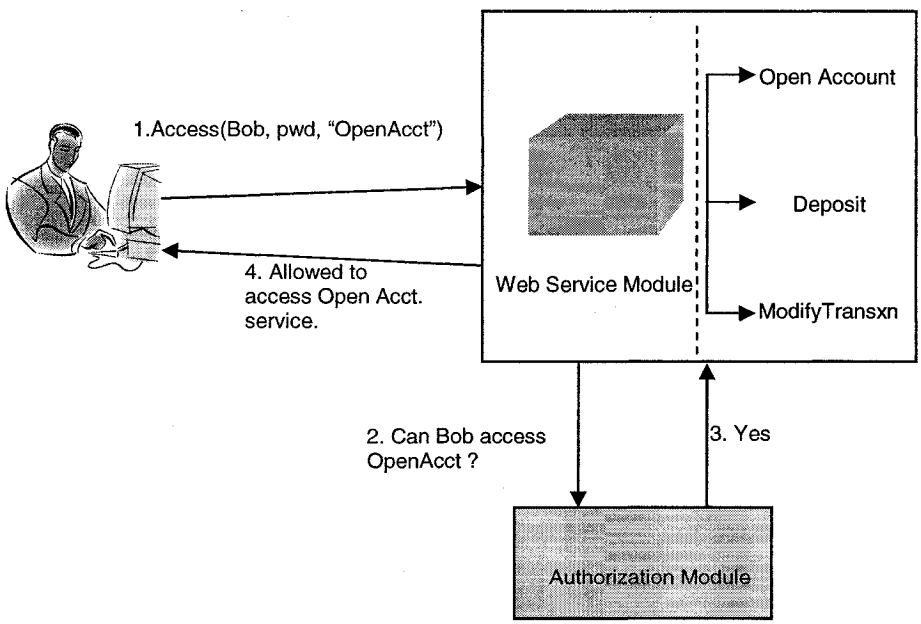


Figure 5: Authorization process for *Open Account* in a banking application

To facilitate designing an authorization strategy, let us break the design process into several steps.

5.1. Choosing an Authorization Strategy

As discussed in Section 2, role based access control, team based access control or resource based access control can be used as an authorization strategy. In this example, role based access control has been used. The basic steps for the implementation of role-based authorization are:

- Authenticate users within the front-end Web application.
- Map users to roles.
- Authorize access to operations (not directly to resources) based on role membership.
- Access the necessary back-end resources.

5.2. Identification of the Participating Roles and Web Services

This phase involves identifying various roles within the bank. These roles maybe based on functionality of users, designation in the hierarchy tree of the banks employees. Given below are the roles and the abbreviations used to represent the roles in the authorization rules and in the database.

(a) **Branch Manager** (BRM): Has the ability to perform any of the functions of other roles at times of emergency and to view all transactions, account status etc.

(b) **Assistant Manager** (ASM): Is responsible for sales, opening accounts and assisting a customer's specific requests.

(c) **Service Manager** (SVM): Acts as a supervisor to the Lead Teller and Teller below him. Ensures daily transactions are carried out efficiently.

(d) **Lead Teller** (LTR): Acts as a supervisor to the teller. He balances cash deposits and withdrawals of transaction performed by tellers under him.

(e) **Teller** (TLR): Inputs and modifies transactions against Customer Deposit Accounts.

(f) **Personal Banker** (PBK): Responsible for home loans, car loans, equities, mortgage.

(g) **Customer Service Representative** (CSR): Opens accounts, handles credit card queries and provides all kinds of assistance to customer.

Figure 6 shows the role hierarchy in a banking organization.

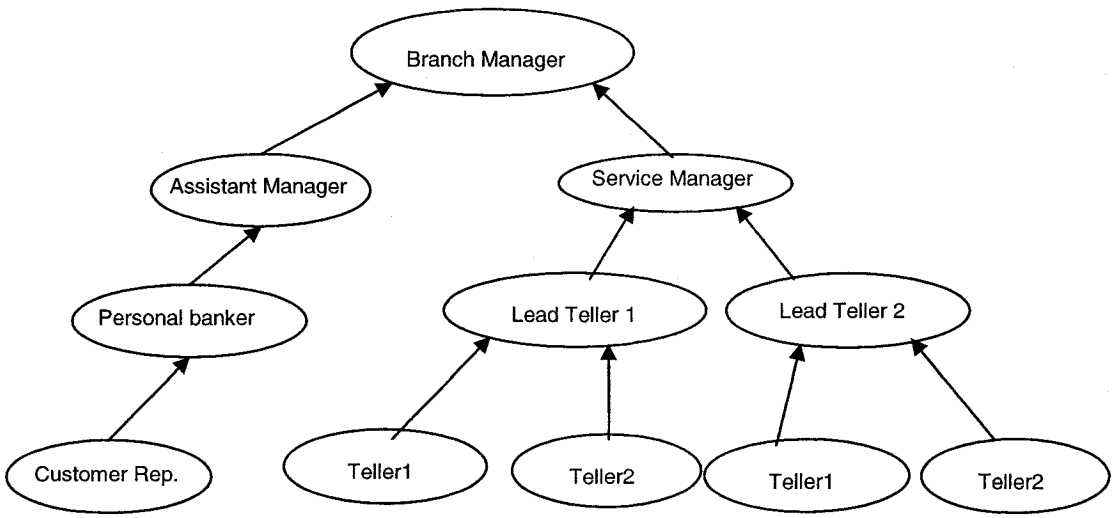


Figure 6: Role hierarchy in iBank

The role identification phase also includes identification of the services the above roles will be given access to.

EDIT_ACCT_INFO: service to modify any customer's account information.

EDIT_TRXN: service to modify an existing transaction.

DEPOSIT: service to make a deposit.

AFTER_HR_ACCESS: service to access the application after usual business hours.

E_CHECK: service allowing customer to use eChecks.

DAILY_TRX: service to access/view daily transaction logs.

GEN_LEDG_REPT: service to generate reports.

MODIFY_LEGD: service to modify ledger reports.

CRT_LOAN_ACCT: service to create a loan account.

MOD_LOAN_ACCT: service to modify loan account.

The above resources are assigned to the participating roles based on their access rights specified in authorization specification input files.

5.3. Formulating the Rules

After the various roles and the services they can access in the bank have been identified, the security administrator formulates the authorization rules after consultation with the stakeholders of the bank. An authorization specification file can consist of macros and authorization rules written in ARSL.

Macros:

The roles can be implemented using macros, as shown below.

```
Roles(x) == "TLR" => Role_TLR(x);
Roles(x) == "CSR" => Role_CSR(x);
Roles(x) == "LOR" => Role_LOR(x);
Roles(x) == "ACC" => Role_ACC(x);
Roles(x) == "AUD" => Role_AUD(x);
Roles(x) == "BRM" => Role_BRM(x);
Roles(x) == "CUS" => Role_CUS(x);
Roles(x) == "PBK" => Role_PBK(x);
```

Here we consider *Role(x)* as user x's role, hence, by design, *Role* would be a column in the employee information table in the database.

Rules:

a. *(forall x) [(Role_TLR(x) OR (Role_CUS(x) AND HasOnlineAccess(x)) => Access(MOD_ACCT_INFO)];*

b. *(forall x) [Role_CUS(x) AND BalanceGT25k(x) => Access(BILL_PAY) AND Access(E_CHECKS)];*

c. *(forall x)[Role_CUS(x) AND (Loan_Acct(x) OR BalanceGT50k(x)) => Access(FREE_TRAV_CHK)];*

d. *(forall x) [Role_ACC(x) OR Role_BRM(x) AND NOT (Role_TLR(x) OR Role_LOR(x) OR Role_CUS(x)) => Access(DAILY_TRX) AND Access(GEN_LEDG_REPORTS) AND Access(MODIFY_LEGD)];*

e. *(forall x) [Role_LOR(x) AND NOT (Role_TLR(x) OR Role_ACC(x)) => Access(CRT_LOAN_ACCT) AND Access(MOD_LOAN_ACCT)];*

f. *(forall x) [(Access(BILL_PAY) AND PMA_Membership(x) => Access(ONLINE_TRADING)];*

Explanation of the Rules:

(a). All users who are Tellers as well as Customers who have online access are allowed to modify their own records

(b). All users who are in the role of Customers who have a balance greater than 25000 have access to *BILL_PAY* and *E_CHECKS* services.

(c). All users who are in the role of Customers who either have a loan account or have a balance greater than 50000 can have access to order free travelers checks.

(d). All users in the role of Accountant and not in the role of Teller or Loan Officer or Customer Service Representative, have access to *DAILY_TRX*, *GEN_LEDG_REPORTS, MODIFY_LEGD.*

(e). All users in the role of Loan Officer and not in the role of Teller or Accountant have access to *CRT_LOAN_ACCT and MOD_LOAN_ACCT.*

(f). All users who have access to bill pay service and have *PMA* membership can access online trading facility.

5.4. Detailed Scenarios

To give a better understanding of the usage of ARSL, three scenarios are described below. These scenarios can occur in the day-to-activities of the bank. The scenarios demonstrate how the proposed authorization model enhances security by providing regulated access to services.

1. Tellers do not have the access rights to open new accounts. Tellers can only accept deposits in the form of cash or check and make withdrawals on behalf of customers.

   Rule: *(forall x) [Role_PBK (x) OR Role_CSR(x) OR*
   NOT Role_TLR(x) => Access (OPEN_ACT)];

   The high-level language code generated for the above rule is:

```
bool AccessOPEN_ACT( int userid )
{
        return (Role_PBK (userid)||
                Role_CSR (userid) || ! Role_TLR (userid));
}
```

2. Senior officers of the bank such as the Branch Manager and Accountant have access to banking services during after hours, others can access only if the current time is within the business hours.

   Rule: *(forall x) [Role_BRM(x) OR Role_ACC(x) OR NOT AfterHours(x)*
   => Access(ACCESS_TIME)];

   In this case, *AfterHours(x)* has been defined as a built-in function. It compares current time with the working hours of the bank.

The high-level language code generated for the above rule is:

```
bool AccessACCESS_TIME( int userid )
{
        return (Role_BRM (userid)|| Role_ACC (userid)||
    ! AfterHours(userid) );
}
```

3. Consider a case where the Teller has already made a deposit on behalf of a customer and suddenly realizes that he made a mistake. In such a situation, only his supervisor can modify the transaction. The supervisor can first view the transactions done and modify an incorrect transaction on behalf of the Teller.

   Rule: *(forall x) [Role_LTR(x) OR Role_SVM(x) OR Role_BRM(x) AND Not Role_TLR(x) => Access (MODIFY_TRX)];*

   In this scenario only certain roles which are higher in hierarchy than the Teller can modify an existing transaction.

   The high-level language code generated for the above rule is:

```
bool AccessMODIFY_TRX ( int userid )
{
        return (Role_LTR (userid)|| Role_SVM (userid)||
            Role_BRM (userid) ||! Role_TLR (userid));
}
```

Security administrators specify authorization rules in a policy file in accordance with the bank's business requirements. In our model of authorization, user context data used in an authorization rule is fetched from the database. A user is a valid user if his data exists in the employee database. This is achieved by using the macros defined above. The assumption here is that any user information that plays a role in evaluating an access rule is fetched from a pre-stored value. To illustrate the above point let us consider an example,

"if user's location is secure then allow him to access ConfidentialDocs".

In this case, the logic to determine whether a location is secure is specified by the Security Administrator using a macro, as shown below,

[Location(x) =="San Jose" => LocationSecure(x)];

The user's location data is fetched from a User object. When the user logs in to the web application using his username and password, after successful authentication, user data is stored in a user object. The user object is populated with values from the database with data relevant to the authorization process in accordance with the business logic. The user data is used for evaluating access rights during authorization. In the above example, to determine whether a user is at secure location or not, the user's location is compared with the value given in the macro i.e., San Jose.

5.5. Implementation Details

The code generated by the compiler is entirely based on the authorization rules and, hence, is application specific. The authorization code consists of an invariant part and a variable part. The invariant part includes code related to database access and other code such as essential functions required to support the actual authorization code but not directly responsible for authorization. These functions do not affect authorization decision and do not change over time. The variable part consists of code generated during the compilation of authorization rules. This code will change as and when changes are made to the input authorization rules. The invariant part is developed in a high-level language, C#, in this case. The rules are written using ARSL, which are compiled to generate code in C#.

Figure 7 shows the data flow diagram of the complete authorization framework. At the inception of the project, during requirement gathering and analysis, security policies need to be formulated. As the application design and development proceed, the list of Web Services to be provided should be decided and the corresponding APIs to access those Web Services should be agreed upon. During the development phase, the invariant part of the authorization module is developed. This will consist of code required to support the authorization code generated, such as database access code, time comparison code. After the application is developed and ready to be deployed, the Security Administrator can specify access rules using ARSL. He can even modify existing rules in accordance with changes in business logic and security environment.

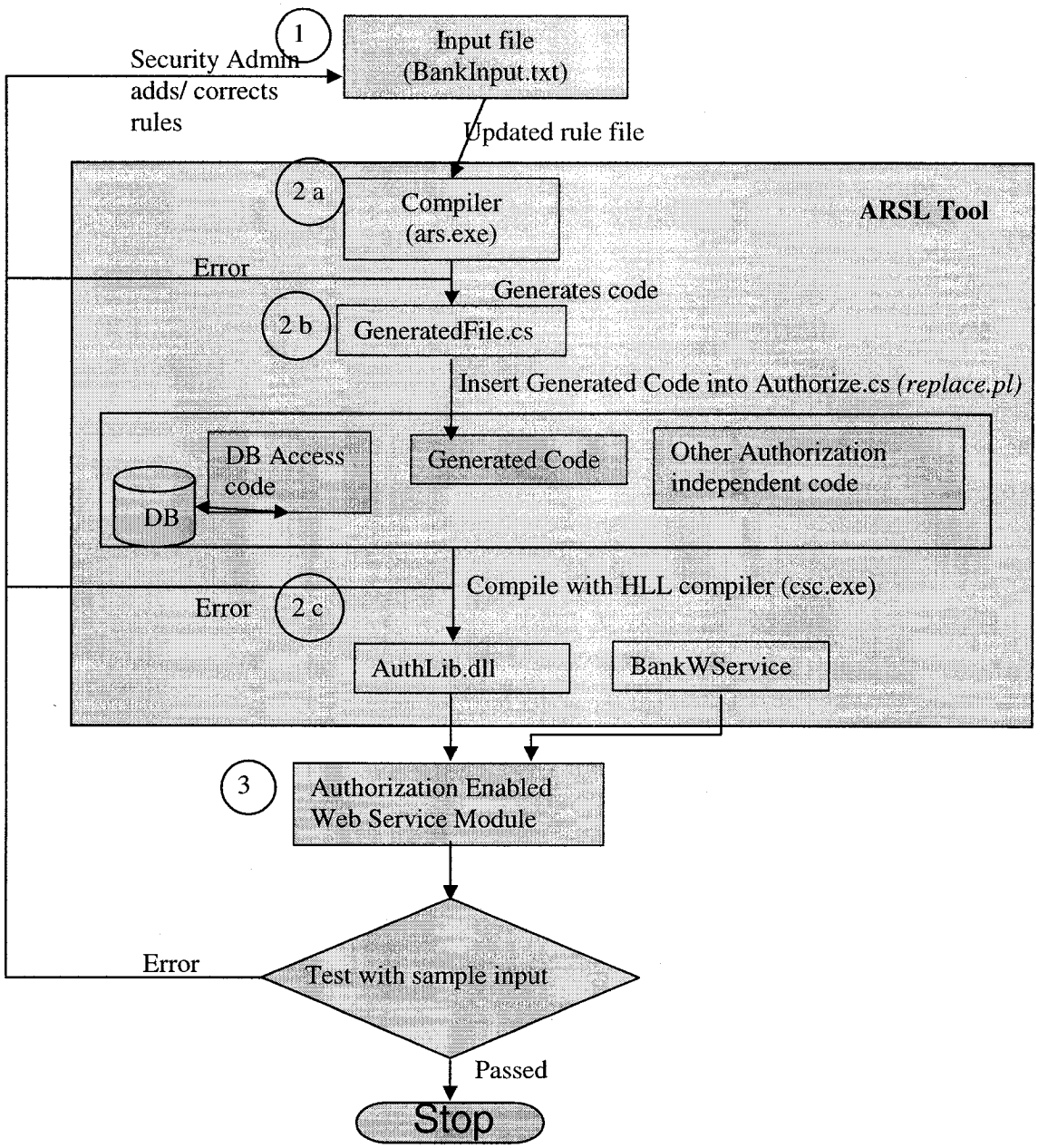
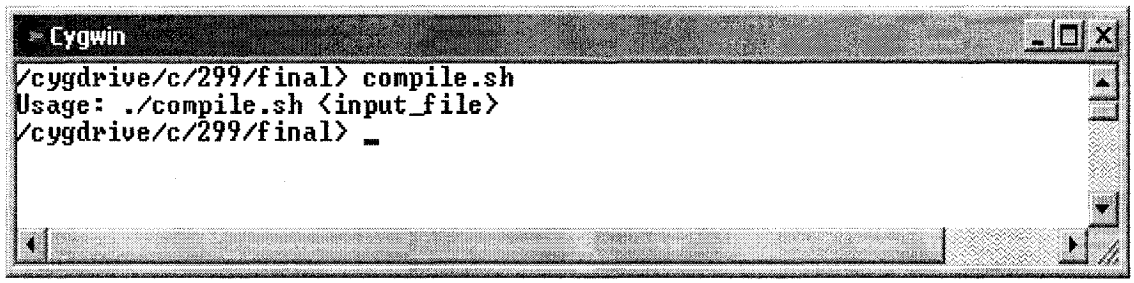


Figure 7: Data flow diagram of the framework

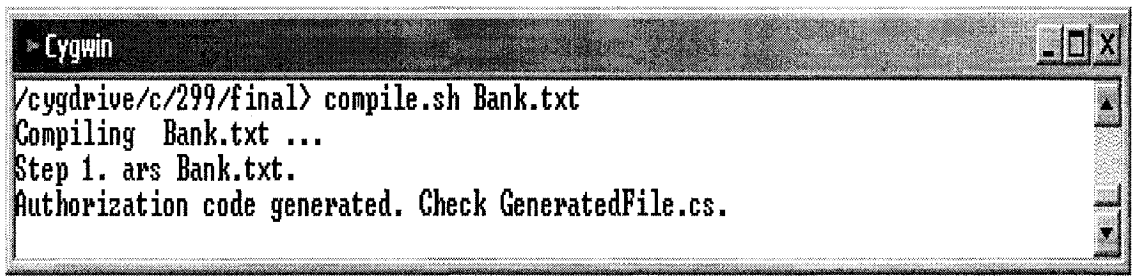
Step 1: Security Administrator adds or modifies authorization specification rules to the rules input file, BankInput.txt.

Step 2: He then runs the shell script *compile.sh* with the above input file as shown in the figure below.



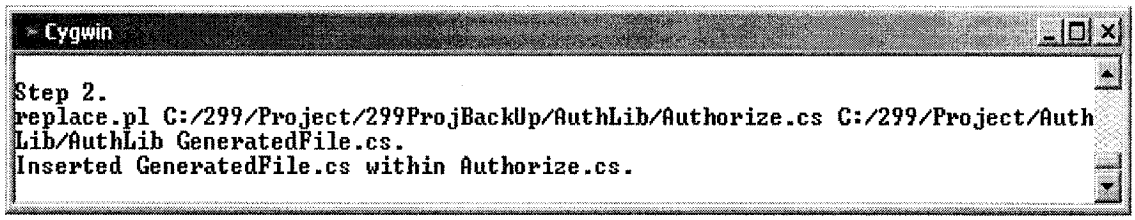
The script performs several functions, such as:

2a. Compiles BankInput.txt using the ARSL compiler, *ars.exe* generating *GeneratedFile.cs*.

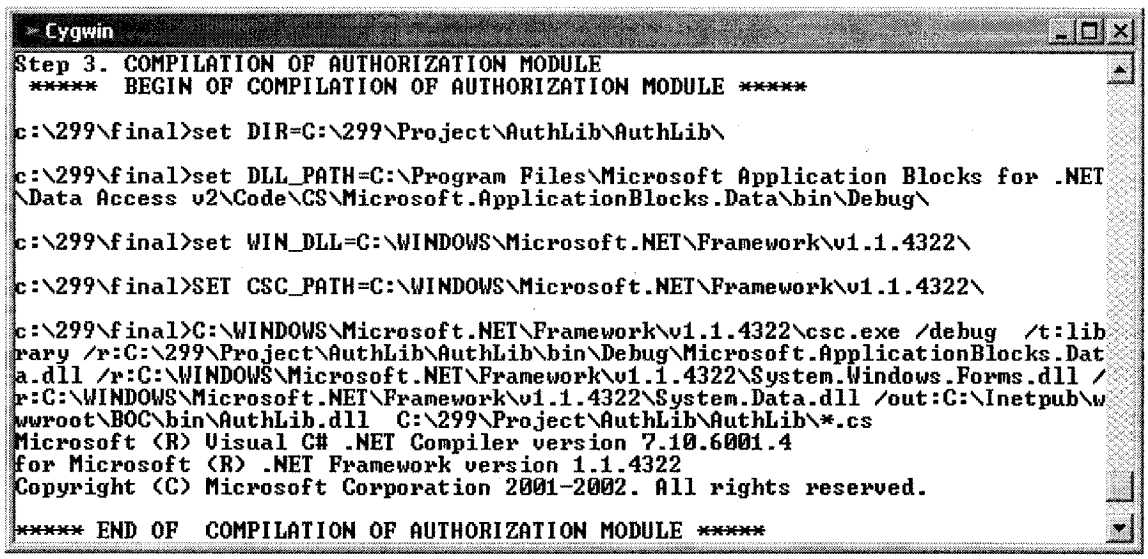


2b. Calls a perl script *replace.pl* to insert the generated code within the Authorization module. As stated before, the authorization module consists of two parts, an invariant part, and a variant part. The invariant part includes code to access database and other essential functions required to support the actual authorization code. The variant part is the code generated dynamically. The perl script is used

to insert the variant part of the code (i.e., the generated code) within the invariant part of code.



2c. Compilation of the authorization module into a dynamic link library.



2d. Linking the authorization Module with the Web Service application.

Step 3: After the Web Service is dynamically linked with the authorization module, a Web Service client can call the services available. In our example, the Bank Client application uses the services provided by iBank Web Service application. Screen shots of the application are shown in the Appendix A.

Error Handling

The compiler is provided with features for error handling. Any error in syntax is caught at compile time and appropriate error message is displayed.

If a Web Service does not exist in the global Service list, the compiler throws an error and exits. The security administrator has to correct the above errors before proceeding with generating authorization code.

In case of conflicts, the authorization module will generate a log file at runtime to inform the Security Administrator. Conflict resolution is described below in detail.

6. Conflict Detection and Resolution in ARSL

In the ARSL model, a security administrator can specify authorization rules without having to change the application code. This is different from models where all the possible access rules are decided from the beginning and access control is coded in the application itself. However, it is not possible to anticipate all access requirements and a dynamic model like the ARSL model is better suited to handle this. The flexibility provided by ARSL to dynamically add authorization rules, is complemented by the conflict resolution strategy built into the model. As rules are added to the input file, multiple rules for one service may result in a conflict. This kind of scenarios arises in enterprises that need to combine various policies in order to implement a complete security infrastructure. In such cases, an authorization model may have multiple rules, where both Access and Deny rules are specified for the same service. This can lead to conflict situations between those rules under certain conditions. ARSL provides users with the capability to specify multiple access rules and deny rules for any service. The significance of this lies in the fact that multiple rules may return different results for a given service for a given user. Conflicts arise when for a given user and a given service, both the Deny and Access functions hold true, thus making it difficult to arrive at a resolution. In such a case, the application must have a way to resolve the conflict and make an unambiguous authorization decision.

6.1. Logical Modeling of Conflicts in Authorization Rules

In a logical system, if one statement is a negation of another statement it results in a conflict in the system. In one logical system, there can be more than one conflict. In an

authorization rule system, every tuple of *access predicate, action* represents one statement. The negation of a statement is another statement, where access predicate remains same but action is negated. For example,

Statement 1:

(forall x) [Role_Teller(x) => Access(Acct_Book)];

Statement 2:

(forall x) [Role_Teller(x) => Deny(Acct_Book)];

Statement 1 and statement 2 have the same access predicate but the actions negate each other. As a result, statement 2 is negation of statement 1.

It can be noted that such negations can be implicit. For example,

(forall x) [Role_Teller(x) => Role_Emp(x)];
(forall x) [Role_Emp(x) => Access(Account_Book)];
(forall x) [Role_Teller(x) AND Country(x) != "USA" => Role_NonUSTeller(x)];
(forall x) [Country(x) != "USA" => NonUS(x)];
(forall x) [NonUS(x) => Deny(EncrySw)];
(forall x) [Deny(EncrySw)=>Deny(Account_Book)];

From the above set of macros and rules, the following statements can be derived.

(forall x) [Role_NonUSTeller(x) =>Role_Teller(x)];
(forall x) [Role_Teller(x) => Role_Emp(x)];
(forall x) [Role_Emp(x) => Access(Account_Book)];
i.e., (forall x) [Role_NonUSTeller(x) => Access(Account_Book)];

(forall x) [Role_NonUSTeller(x) = > NonUS(x)] ;
(forall x) [NonUS(x) => Deny(Account_Book)];
i.e., (forall x) [Role_NonUSTeller(x) => Deny(Account_Book)];

Thus, it can be observed that the original access rules and macros may not directly show the conflicts. However, using logical derivations, new access rules can be derived

which can show the conflicts in the system. This type of conflict is known as implicit conflict.

With the above understanding, we can formulate necessary and sufficient condition for existence of conflict in an authorization system.

For a given service R, there is a conflict if and only if there exists

1. Access rule Ai □ A, where A is the set of Access rules for R
2. Deny rule Dj □ D, where D is the set of Deny rules for R
3. User x, such that the logical expression [Ai(x) AND Dj(x)] is true

The proof of the above formulation is as follows:

If statements 1-3 hold good, then for user X we have both access and denial for service R, which generates a conflict. Conversely, if there is a conflict for service S then, there is at least one instance of a user, for whom there is an access conflict. This implies at least one instance of access rule A, such that Ai(x) is true and at least one instance of deny rule D, such that Dj(x) is true i.e. Ai(x) AND Dj(x) is true.

6.2. Ways of Resolving Conflicts in Authorization Rules and Trade offs

6.2.1. Restrict New Users such that Conflict is not possible

While adding a new user X, ensure that there is no conflict for X. This will need evaluating all the access rules and denial rules for user x for all the services. In case of a conflict this means, either not adding the new user x or modifying the conflicting rules to ensure there is no conflict for the new user x and for the existing users.

6.2.2. Restrict Authorization Rules such that Conflict is not possible

While adding a new access authorization rule find the subset of users A, for which this authorization rule holds true and then evaluate all the denial rules for the subset of users A. If any denial rule holds true for any user in A, do not add the new access authorization rule. This implies the security administrator has to modify the authorization rules so that the new rules do not result in a conflict. This may involve merging the new rule with existing rules that cause the conflict in order to avoid the conflict. It an actual system, new users, new rules, new services may be added.

In order to ensure conflict resolution by construct both the above solutions have to be used. The overhead for adding a new user or new rule is high, but it helps in avoiding any conflict testing and resolving at runtime.

6.2.3. Restricting Authorization Rules to Only One Type (access rules or deny rules, but not both)

In case of only access rules for a service, express all the denial rules for that service in terms of access rules and vice-versa. In any authorization system, more than one access rule for a service means logical ORing of those access rules. Converting denial rules involves replacing each denial rule by negation of its predicate and ANDing that with the predicates of all access rules for that service. The choice of only access rule or only denial rule for a service will depend on the conflict resolution policy for that service. In case the resolution is for denial then only access rule should be allowed and vice-versa. Since more than one access rule means logical ORing of all the access rules, without loss of generality the system can be restricted to one authorization rule per service to avoid conflict.

The following example explains conflict resolution by restricting authorization rules for a service to either access rules or deny rules.

Input: Service R

Rules: A1(x) => Access(R)
A2(x) =>Access(R)
...
Am(x) => Access(R)
D1(x)=Deny(R)
D2(x)=Deny(R)
...
Dn(x)=Deny(R)

The table below illustrates the various ways in which conflicts can be resolved by restricting authorization to either access or deny rules. It shows how the way differs depending on whether the resolution is towards access or deny.

Table 1: Conflict Resolution by Rules Modification

	Resolve Conflict – Deny	**Resolve Conflict – Access**
Deny Rules only	NOT (A1(x) OR A2(x) …OR Am (x)) => Deny(R) D1(x) => Deny(R) D2(x) => Deny(R) … Dn(x) => Deny(R)	D1(x) AND NOT(A1(x) OR A2(x) …OR Am (x)) => Deny(R) D2(x) AND NOT(A1(x) OR A2(x) …OR Am (x)) => Deny(R) … Dn(x) AND NOT(A1(x) OR A2(x) …OR Am (x)) => Deny(R)
Single Deny Rule only	(D1(x) OR D2(x) … OR Dn(X)) OR (NOT (A1(x) OR A2(x) …OR Am (x))) => Deny(R)	(D1(x) OR D2(x) … OR Dn(X)) AND (NOT (A1(x) OR A2(x) …OR Am (x))) => Deny(R)
Access Rules only	A1(x) AND NOT(D1(x) OR D2(x) … OR Dn(x))=> Access(R) A2(x) AND NOT(D1(x) OR D2(x) … OR Dn(x))=> Access(R) … Am(x) AND NOT(D1(x) OR D2(x) … OR Dn(x))=> Access(R)	NOT(D1(x) OR D2(x) … OR Dn(x))=> Access(R) A1(x) => Access(R) A2(x) => Access(R) … Am(x) => Access(R)
Single Access Rule only	(A1(x) OR A2(x) …OR Am (x)) AND NOT(D1(x) OR D2(x) … OR Dn(x)) => Access(R)	(A1(x) OR A2(x) …OR Am (x)) OR NOT(D1(x) OR D2(x) … OR Dn(x)) => Access(R)

6.2.4. Priority Based Conflict Resolution Mechanism

In general, in an authorization system different services may have different resolution criteria. Irrespective of resolution criteria, it may be preferable to specify either access or deny rules. The solution described in Section 6.2.3 can work if:

- The conflict resolution for a service can be pre-determined to either access or deny.
- All the rules have the same priority.

In actual authorization systems, there may be exceptions and these constraints may not hold good for all services.

For example, an online shopping website may start with the following simple rule:

Rule 1: If the user has a valid credit card, authorize him to purchase item.

(forall x)[CreditCard == "INVALID" =>Deny(Item)];

Rule 2:

(forall x)[Prescription == "Item" => Access(Item)];

Depending on a country's law, other authorization rules are added.

Rule 3:

(forall x)[Country =="USA" AND Age < 18 AND Item == "Liquor" => Deny(Item)];

Rule 4:

(forall x)[Country =="Germany" AND Age < 21 AND Item== "Liquor" => Deny(Item)];

The conflict resolution of these rules cannot be simplified to access or deny because for a shopping chain, the payment is mandatory and rule 1 has the highest priority. Among the other rules, rule 4 has the next level of priority while rules 2 and 3

have the same next level of priority. If the conflict is between rule 1 and rule 4, then rule 1 takes precedence, resulting in denial. If the conflict is between rules 2 or 3 and 4 then rule 4 takes precedence, resulting in allowing access. Therefore, the solution in Section 6.2.3 may not be applicable in this case. As discussed in (Anderson 53-60; Adam, Mitchell, Rosenstein 45-46), a language such as IBM's Enterprise Privacy Authorization Language (EPAL) ("Enterprise") may use the order of the rules to determine applicability of a rule for a given request. The previous solution can be modified to evolve into a conflict resolution mechanism based on priority of rules. It may be noted that an authorization system without priority among the rules is a special case of an authorization system with priority among the rules where, all the rules have the same priority.

Algorithm

Inputs: A set of authorization rules given in the order of priority
Output: A single authorization rule resolving conflicts based on the priority
Current Predicate = Predicate of Rule n
Current action = Action of Rule n
For i =n-1 to 1 DO

```
BEGIN
        IF Action of Rule i==Current Action
        THEN
                Current Predicate = (Predicate of Rule i) OR (Current Predicate)
        ELSE
                Current Predicate = NOT (Predicate of Rule i) AND ( Current Predicate)
        ENDIF
END
IF (Current action = ="Deny")
THEN
        Current Predicate = NOT ( Current Predicate)
        Current Action = "Access"
ENDIF
```

Writing the rules in order of priority, we have,

Rule 1: *(forall x) [CreditCard == "INVALID" =>Deny(Item)];*

Rule 2: *(forall x) [Prescription == "Item" => Access(Item)];*

Rule 3: *(forall x) [Country =="USA" AND Age < 19 AND Item == "Liquor" => Deny(Item)];*

Rule 4: *(forall x)[Country =="GERMANY" AND Age < 21 AND Item == "Liquor" => Deny(Item)];*

Rule 5: *(forall x)[TRUE => Access(Item)];*

Rule 5 is the default rule, applied when none of the predicates is true.

Using the above algorithm, the resultant authorization rule is

i=4:

NOT (Country(x) =="Germany" AND Age(x) < 21 AND Item(x) == "Liquor") AND True) => Access(Item)

i=3:

NOT (Country(x) =="USA" AND Age(x) < 18 AND Item(x) == "Liquor") AND (NOT (Country(x) =="Germany" AND Age(x) < 21 AND Item(x) == "Liquor") AND True))=> Access(Item)

i=2:

(Prescription(x) == "Item") OR (NOT (Country(x) =="USA" AND Age(x) < 18 AND Item(x) == "Liquor") AND (NOT (Country(x) =="Germany" AND Age(x) < 21 AND Item(x) == "Liquor") AND True)))=> Access(Item)

i=1:

NOT (CreditCard(x) == "INVALID") AND ((Prescription(x) == "Item") OR (NOT (Country(x) =="USA" AND Age(x) < 18 AND Item(x) == "Liquor") AND (NOT (Country(x) =="Germany" AND Age(x) < 21 AND Item(x) == "Liquor") AND True))))=> Access(Item)

As the default action is Access, the resultant rule is same as the rule in the last iteration.

NOT (CreditCard(x) == "INVALID") AND ((Prescription(x) == "Item") OR (NOT (Country(x) =="USA" AND Age(x) < 18 AND Item(x) == "Liquor") AND (NOT (Country(x) =="Germany" AND Age(x) < 21 AND Item(x) == "Liquor") AND True))))=> Access(Item)

6.3. Advantages of the Proposed Algorithm

From the previous example, it may be observed that the resultant single predicate is a simple logical expression using only three logical operators- NOT, AND, and OR. The expression is designed so that it can be evaluated from left to right and evaluation can stop as soon as any predicate is true.

Merging the predicates of multiple rules makes it suitable for exploring chances of parallelization. For example, a Very Long Instruction Word (VLIW) processor having three logical processing units can process three terms of the final predicate in parallel and make the evaluation three times faster. As all the predicates involve data of a single user, the data can be fetched from the database in a single query.

In general, the operations involved can be divided into two steps

- Fetching data for a user
- Evaluating the applicable logical predicates for the user

As these two activities are sequential (i.e., the evaluation has to wait till the data is fetched), these cannot be parallelized. However, in real time since this operation is done for more than one user it is a good case for using pipelining to make it even faster. In a simple scenario, access evaluation for two users can be done in a pipeline. When the *i*th user's data is being fetched from the data from the database the *(i-1)*th user's data maybe evaluated. As a result, both data fetching and evaluation can be done in parallel. If the task is further subdivided, for example, if fetching of data can be subdivided into fetch data, transmit data, receive data, decode data, filter data, then the pipelining can have even more number of stages resulting in increased throughput. This has been possible

only because we have been able to combine all the rule predicates into a single predicate, and achieve isolation between data retrieval and evaluation.

6.4. Choosing a Conflict Resolution Mechanism for ARSL

As discussed in the previous sections, the solutions described in Sections 6.2.1 and 6.2.2 have drawbacks. They have overheads for adding new rules and new user. The third solution given in Section 6.2.3 is applicable only when the constraints are satisfied. The fourth solution from Section 6.2.4 is a superset of the third solution and has no known drawback. Hence, the fourth solution is our choice for implementation in ARSL.

6.5. Examples of Conflicts in ARSL

As described earlier, ARSL rules are of the form:

Predicate => Access(service);
Predicate => Deny(service);

Where the LHS represents the conditions for which the actions on the RHS hold true. To provide a better understanding, some additional terms are defined below.

Predicates: The terms on the LHS of rules representing conditions. These can be macros or other Access () or Deny() functions.

Actions: Can be either Access(service) or Deny(service).

Access Predicate Set (service): Set of predicates for all Access() rules for a service. When this set has more than one predicate the result is logical OR of all the predicates. This set can be NULL.

Deny Predicate Set (service): Set of predicates for all Deny () rules for a service. When this set has more than one predicate the result is logical OR of all the predicates. This set can be NULL.

Conflict: If for a given user data and a service, results of both Access Predicate Set() and Deny Predicate Set() are true then there is a conflict. The resolution mechanism in case of a conflict should be predefined. In the examples given below, the conflict resolution mechanism chosen allows Deny rule to override the Access rule.

The following example will explain the above terms. There are four rules for the service Acct_Book.

(forall x) [Role_Teller(x) => Access(Acct_Book)];
(forall x) [Role_Manager(x) => Access(Acct_Book)];
(forall x) [Role_Customer(x) => Deny(Acct_Book)];
(forall x)[Role_LoanOfficer(x) => Deny(Acct_Book)];

In the above set of rules, the terms defined above are as follows:

Service: {Acct_Book}

Predicates :

{Role_Teller(x)},
{Role_Manager(x)},
{Role_Customer(x)},
{Role_LoanOfficer(x)}

Actions:

{Access (Acct_Book)}, {Deny (Acct_Book)}

Access Predicate Set (Acct_Book): {Role_Teller(x), Role_Manager(x)}

Result of Access Predicate set (Acct_Book): Role_Teller(x) OR Role_Manager(x)

Deny Predicate Set (Acct_Book): {Role_Customer(x), Role_LoanOfficer(x)}

Result of Deny Predicate Set(Acct_Book): Role_Customer(x) OR

Role_LoanOfficer(x)

If both Access Predicate Set (Acct_Book) and Deny Predicate Set (Acct_Book) are true for a given user, then there is a conflict.

For example, if John Doe is promoted from the role of a Teller to a Loan Officer, but has not yet been removed from the database as a Teller, then a request to access Acct_Book will generate a conflict

The table below shows the decision table showing conflict scenarios.

Table 2: Decision Table for Access(Acct_Book)

Predicate1 Teller	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
Predicate2 Manager	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
Predicate3 Customer	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
Predicate4 Loan Officer	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Access				X				X				X				
Deny													X	X	X	X
Conflict	X	X	X		X	X	X		X	X	X					

6.6. Implementation of Conflict Resolution in ARSL

ARSL implementation includes both conflict detection and conflict prevention mechanism. Conflict detection technique is chosen by using a special ARSL compiler flag. Using the '–D 'compiler option will generate code for conflict detection. The conflict detection mechanism evaluates all the access and deny rules for a service and for a given user. When a conflict occurs, a log file is generated specifying the rules that caused the conflict and the user for whom the conflict occurred and the resolution chosen. This approach is a debug approach to validate the accuracy of the authorization rules. The conflict prevention mechanism is a preferable method to overcome conflicts and is the default compiler option. For example, for the input given earlier a log file is generated.

In this approach, the administrator expresses the conflict resolution policy in ARSL, using the following two language semantics.

- For a given service, if more than one rule is specified, then the order of priority of the rules is same as order of occurrence of those in rule description file i.e. if $rule_m$ for service S precedes $rule_n$ in the rule description file, then priority of $rule_m$ is higher than that of $rule_n$.
- At the end of the rule description file, default authorization rule must be specified for each service using TRUE predicate. For example, if the default authorization rule for a service S1 is deny, then the default authorization rule for S1 is "true => Deny(S1)", whereas if the default authorization rule for a service S2 is access, then the default authorization rule for S2 is "true => Access(S2)". By putting these default rules at the end of the authorization specification file, we ensure that the priorities of these rules are the lowest and these will be used only when none of the other access rule applies.

ARSL implements this conflict resolution policy by using the algorithm described in Section 6.2.4 by reducing multiple access rules for a resource to a single access rule.

6.7. Comparison between using Priority and no Priority among Authorization Rules

This section presents a comparison between two approaches of conflict detection. As mentioned before conflicts occur when there are multiple authorization rules (Access or Deny rules) for a given resource and at least one Access and one Deny rule return true. The first approach is to allow the conflicts to occur at runtime and let the system administrator take a decision after the conflict occurs. In this approach, the priority of rules is not important. The second approach is the usage of a priority based rule specification to avoid conflicts among rules for a given resource, as implemented in

ARSL. In order to avoid conflicts, the second approach is preferable. This approach relieves the security administrator from having to take a decision when conflicts occur; rather the decision is already made at the time of rule specification.

A runtime comparison between the above approaches is given below. The same set of input rules is run in both cases and their execution time was collected. As can be seen from the graph below, the execution time taken by a priority-based rule specification is significantly lower than using a non-priority based rule specification. In a priority-based approach, the authorization function call returns as soon as one of the rules is true. However, in a non-priority based approach, all the rules have to be evaluated in order to arrive at a decision. This adds to the execution time.

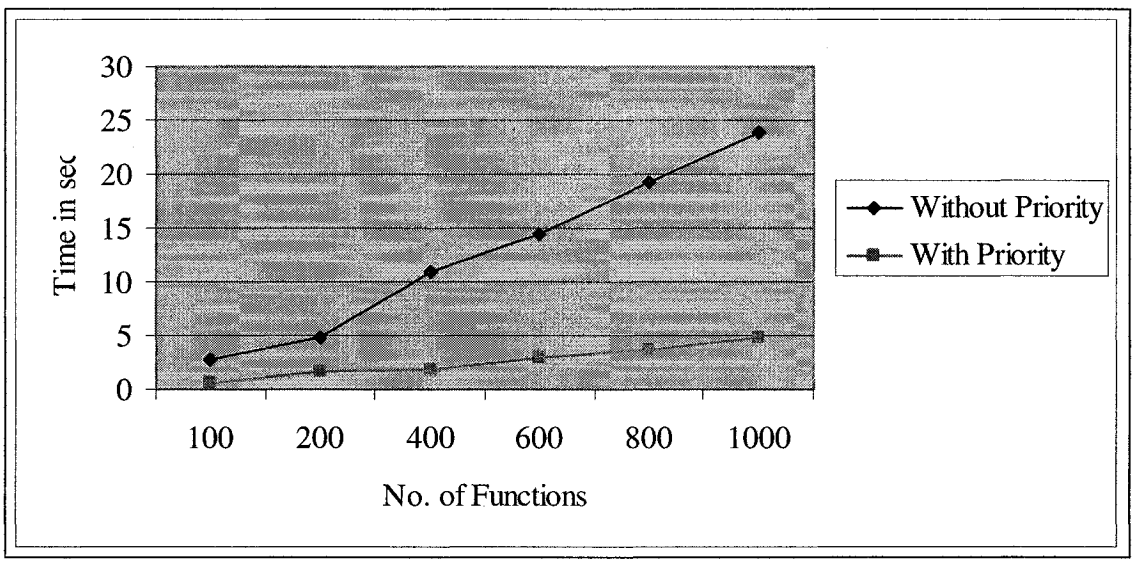


Figure 8: Comparison of execution time between rules specified with and without priority

7. Features of ARSL

Simplicity- It has few constructs, which are English-like. Security administrators can describe an access-control policy once, without having to rewrite it numerous times in different application-specific languages. The simplicity of ARSL helps in avoiding confusions and errors. It also makes it easy to implement a translator that can convert the rules to authorization software.

Maintainability- In most scenarios, authorization requirements are constantly evolving and have similarities. Macros in ARSL facilitate reuse of components of previous authorization rules. This helps in quick development of authorization rules, minimizing errors and facilitating easy maintenance, any change in a single component does not require changes in more than one place.

Completeness- The language is complete in that all authorization requirements can be expressed in this language.

7.1. Advantages of Using a Framework and ARSL

The main advantages of using a special purpose language to express authorization policies and to generate code from those policies are:

Encapsulation: This approach decouples the authorization rules from the Web Service implementation thus encapsulating authorization. The rules are the same irrespective of the platform used. Hence, changes to the rules do not require modification to the code, a costly and error-prone process.

Ease of expression: It is easier to express authorization rules in a language based on predicate logic, which is well known for its knowledge representation capabilities.

Single point of access: All the security information can be concentrated in a centralized point of control, thus preventing any inadvertent changes and errors.

Scalability: Delegated administrators can manage a single set of policies. For example, the corporate security administrator can assign administration roles to separate departments or sub-groups.

Manageability: If a policy needs to be modified it is automatically available to all applications and Web Services using it since, a change in the rule will result in a change in the authlib.dll only, thus minimizing the complexity of propagating and managing the change.

Conformance: Rule-based authorization model will help in conforming to established company policies.

Reusability – Developers can reuse existing code.

8. Comparison of XACML and ARSL Features

1. One of the main features of XACML is to allow defining multiple rules. In such a scenario, different policies can generate conflicting rules. XACML provides rule-combining algorithms to resolve such conflicts to arrive at one outcome per policy. For example, the Deny Overrides Algorithm specifies that, if any evaluation returns Deny, or no evaluation permits, the result is also Deny (Simon and Moses). Currently, ARSL does not allow having multiple access rules for a resource. ARSL can be extended to allow defining multiple rules by providing appropriate conflict resolution mechanism, similar to the rule-combining algorithm in XACML. These combining algorithms used to build increasingly complex policies, helps XACML policies to be distributed.
2. XACML provides a generic approach to access control implementation. This means that rather than providing access control for each particular environment or each specific kind of resource separately, developers can use XACML in any environment (Simon and Moses). After a policy is written, it can be used by different kinds of applications, and when one common language is used, policy management becomes much easier. As part of the future work, ARSL can generate code based on parameters specifying the environment.
3. XACML is distributed, since a policy can refer to other policies kept in arbitrary locations. This helps in maintainability, rather than having to manage a single policy, different people or groups can manage separate sub-policies as appropriate. XACML can correctly combine the results from these different policies into one decision

(Simon and Moses). Since ARSL depends on the precompiled authorization module, having distributed policies is not possible. This is not a major disadvantage, since in real world distributed policies are not practical because of security and efficiency reasons.

4. The XACML standard language supports a wide variety of data types, functions, and rules about combining the results of different policies (Simon and Moses). ARSL can be extended to include a wide variety of data types.
5. XACML is intended primarily to be generated by tools, since its verbose syntax makes it hard to read and tedious to edit for other than very simple policies. Other than Sun's XACML implementation in Java, commercial tools are yet to be developed. As a result it is difficult for Security Administrator to generate the authorization policies without help of experts in XACML (Simon and Moses). In comparison, ARSL has a simple language syntax, which can be easily used by the Security Administrator.

Table 3 gives a comparison of features present in XACML and ARSL. The future work column indicates whether a given feature available in ARSL can be implemented in ARSL.

Table 3: Summary of Features of XACML and ARSL

Serial No.	Features	XACML	ARSL	ARSL* (future work)
1.	Multiple rules can be defined for a resource, thus generating conflicting rules that can be resolved using rule-combining algorithms.	Yes	No	Yes. ARSL can be extended to allow defining multiple rules by providing appropriate conflict resolution mechanism, similar to the rule-combining algorithm in XACML.
2.	Generic approach to access control implementation. After a policy is written, it can be used in different environments.	Yes	No	Yes. ARSL can be extended to generate code based on parameters specifying the environment.
3.	Distributed policies i.e., a policy can refer to other policies kept in arbitrary locations.	Yes	No	No. Since ARSL depends on the precompiled authorization module, having distributed policies is not possible. This is also advantageous from a security and efficiency perspective.
4.	Variety of data types, functions, and rules about combining the results of different policies.	Yes	Yes (Subset of XACML data types.)	Yes. ARSL can be extended to include a wide variety of data types.
5.	Ease of language use.	No	Yes	Yes. ARSL has a simple language syntax which can be easily used by the Security

				Administrator. XACML is a verbose language and requires a tool to generate code.
6.	A method for basing an authorization decision on attributes of the subject and resource.	Yes	Yes	Yes
7.	Logical and mathematical operators.	Yes	Yes (Subset of XACML operators.)	Yes
8.	Method for identifying the rule that applies to a given action, based upon the values of attributes of the subjects, resource and action.	Yes	No	Not required. Since ARSL converts authorization requests to function calls based on resource name.
9.	An abstraction-layer that insulates the policy-writer from the details of the application environment.	Yes	Yes	Yes
10.	Policies based on subject and resource attributes.	Yes	Yes	Yes

9. Performance Evaluation

This section evaluates the performance and scalability of the authorization framework and the compiler developed for parsing authorization. One of the important features of the proposed authorization framework is the ability to allow real-time updates to the authorization rules. This implies that the time required to integrate a change in authorization policy is insignificant compared to the running time of the service.

9.1. Time versus Number of Functions in a Single Access Rule

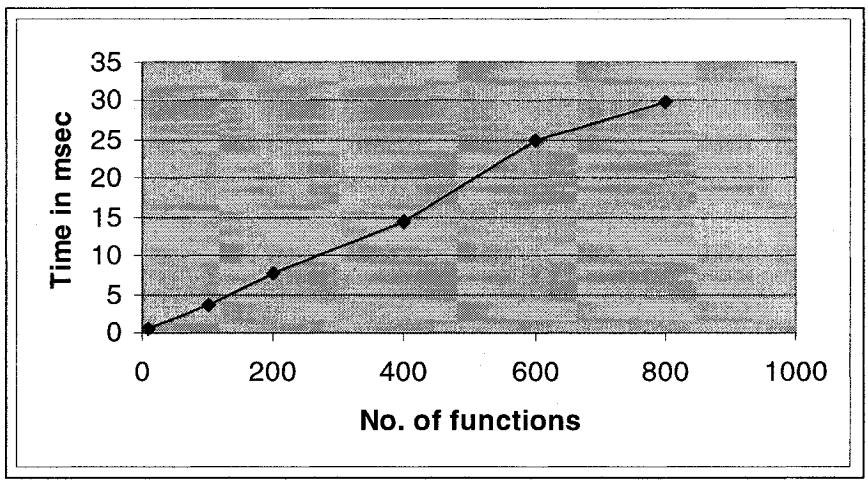


Figure 9: Graph showing code generation time versus number of functions in each rule

The graph above shows the time in milliseconds versus number of functions in a single access rule. The above graph was plotted using several input files. Each input file had a single authorization rule with varying number of constituent functions, macros and other access functions. The graph shows that although the number of constituent functions was increased from 200 to 800 the time taken to generate the authorization code is insignificant compared to the number of functions that constitute a single rule. As can

be seen, with number of functions approaching one thousand, the time taken is less than a second. Hence, modifying existing authorization rules does not carry a huge overhead in terms of time to generate new authorization code. Once the code is generated, compiling the code to generate the AuthLib.dll and linking it with the Web Service application has been automated using a batch program. Thus, once the security administrator modifies the authorization rules, rest of the steps of implementing the new authorization rules is just a matter of running a batch program. This adds a huge benefit in today's fast changing business policies and does not require the security administrator to be program savvy.

9.2. Time versus Number of Rules in a Single Input File

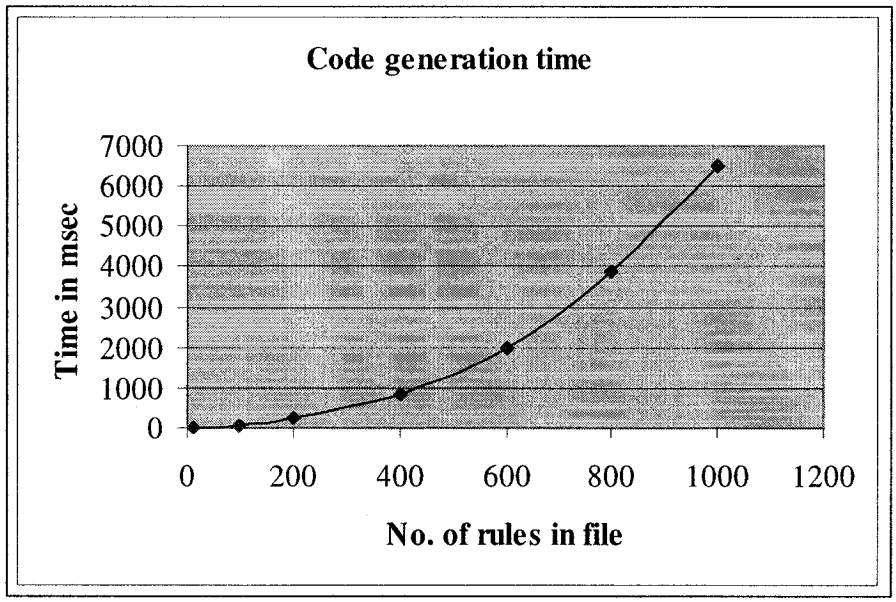


Figure 10: Time versus number of rules in an input file

The graph above shows the time taken by the ARSL compiler, ars.exe to generate authorization code in high-level language. The graph is plotted between the number of rules in a sample input file and the time taken to generate the code.

9.3. Time versus Number of Predicates per Rule

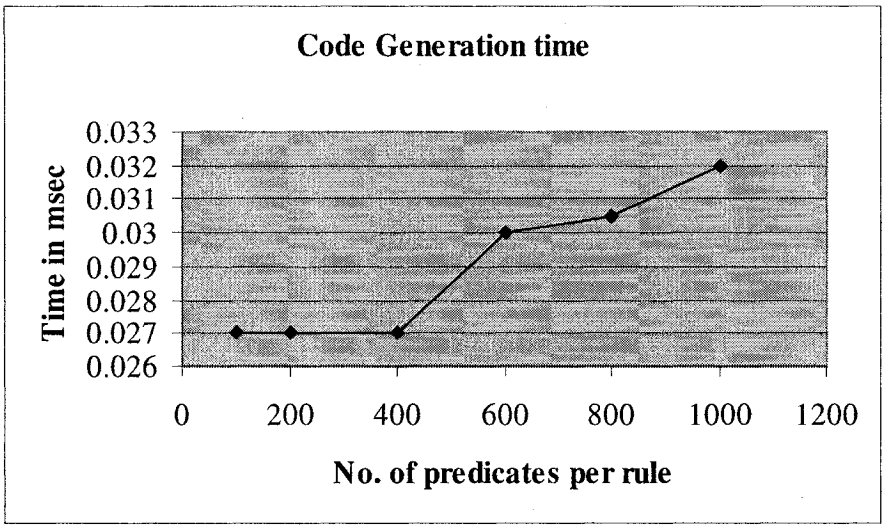


Figure 11: Time versus number of predicates per rule

The above graph shows the time taken to call a given authorization function at runtime. The data was obtained by timing the calls to complex access rules. The number of functions in each complex access rule varied from 200 to 1000 functions each. The aim was to show that although a single call to a complex access rule from a Web Service will result in hundreds of function calls, the time it takes is quite insignificant.

9.4. Execution Time Comparison between ARSL and XACML

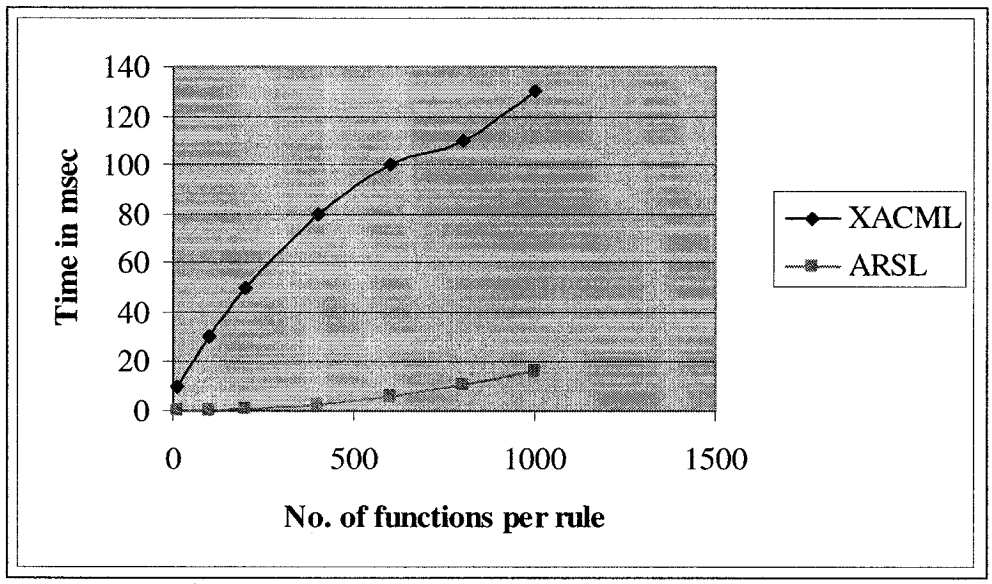


Figure 12: Number of functions in each rule versus code execution time using XACML and ARSL

The above graph shows the execution time taken by an implementation of similar policies using ARSL and XACML. The plot for ARSL shows the number of functions per rule versus the execution time for authorization. The plot for XACML shows the number of user attributes per policy versus the execution time for authorization.

From the above graph, it can be concluded that the execution time taken by ARSL code is about 100 times faster than the time taken by XACML code. The difference in execution time between ARSL and XACML can be attributed to the differences in the implementation of authorization. In case of ARSL, authorization is implemented in executable code whereas in case of XACML, authorization is implemented by data comparison. The XACML request data is parsed and compared with the XACML policy document. This comparison has to be done in real-time for every request resulting in

runtime inefficiency. With increase in the number of attributes of the user, time taken in parsing increases significantly, as seen from the graph of XACML.

Figure 13 and 14 show data flow diagrams for ARSL and XACML respectively. In ARSL model, the compiler parses authorization rules and generates high-level language code (hll). A high-level language compiler, such as *cs*, compiles and generates the authorization module library. Whenever a user requests access to a service, the Web Service calls the access function for the requested service with the user's attributes as parameters. In comparison, XACML uses a request based on the user's attributes at the Policy Enforcement Point (PEP). The PEP sends the request to the Policy Decision Point (PDP) for evaluation. In this case since data is in the form of XML, first parsing has to be done at the PDP to find a matching policy for the request and then to compare the request with the policy (Simon and Moses).

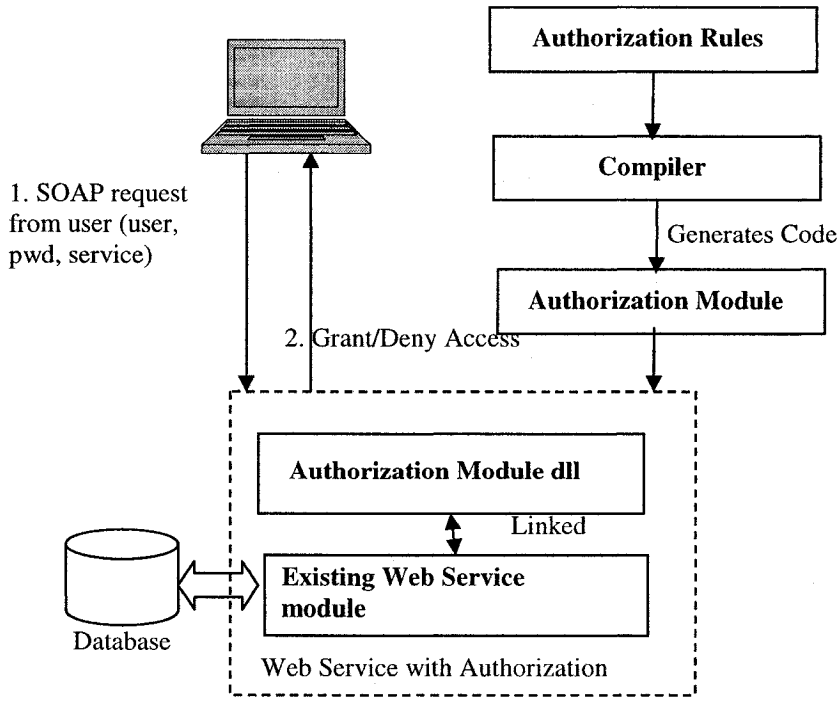


Figure 13: ARSL: data flow diagram

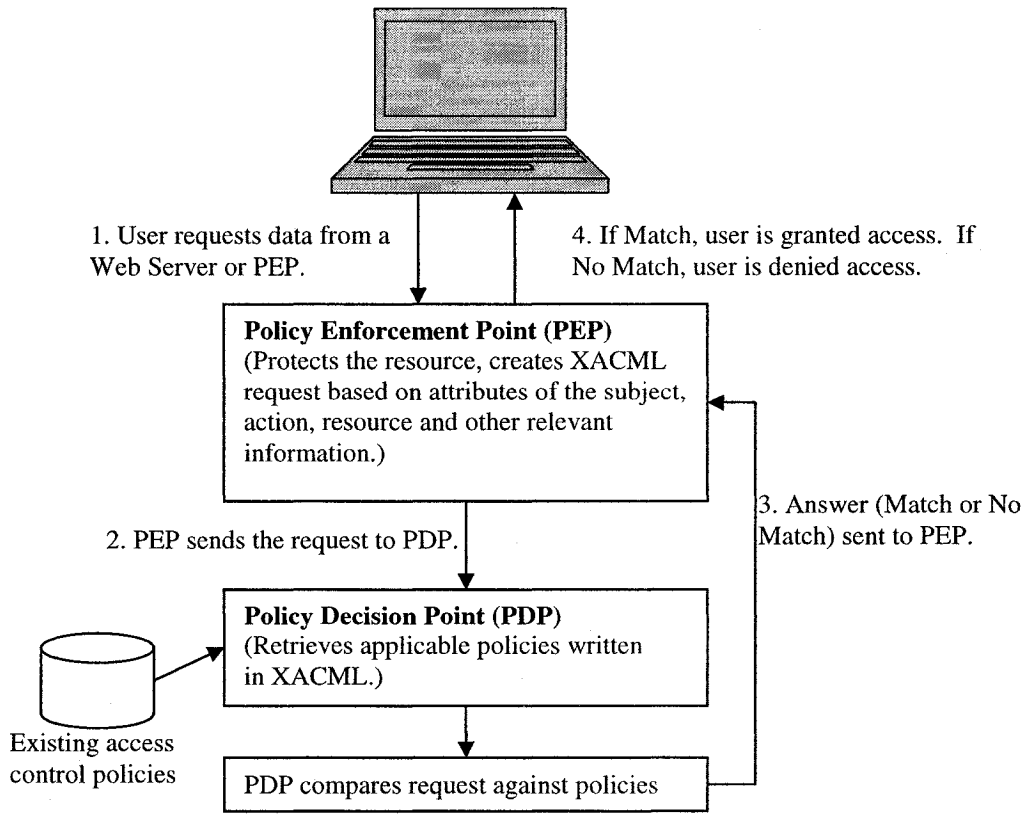


Figure 14: XACML: data flow diagram

10. Open Issues and Future Work

The main goal of this thesis work is to automate and componentize security and authorization specification for Web Services. The current implementation shows an authorization framework using the Authorization Rule Specification Language. This model will work fine only when users do not give erroneous inputs to the translator. Since the language is simple, chances of errors are less. Error recognition and correction are some of the areas that can be developed in future. Errors can be syntactic (easy to catch in the translator), and semantic or even due to wrong usage. User can inadvertently give permission (or deny permission) to an unauthorized request. Therefore, there should be a way to help a user in simplifying his rule specification, and avoiding such mistakes. There should be a facility for user to test sample test cases to validate his authorization specification and to verify the exhaustiveness of these test cases. As part of the future work, this language can be extended to include more features and can be based on different authorization mechanisms. Date and time comparison requires complex rules and can be a new feature to be added in future. Current implementation in our framework allows these features with the help of macros using floating-point numbers or string comparisons. Since time comparison rules can be complex, these macros can be implemented in the fixed part of the authorization module that would be easier to maintain in future. Various authorization models such as team-based, group based authorization models can be implemented depending on the business needs. Authorization input can be huge for certain applications where the time constraint can be as less as few minutes to modify the rules, generate the authorization module, and

integrate with the Web Service application. So common techniques of divide and conquer (parallelization) can be explored. Future work will involve extending the authorization framework to other platforms. The framework can be modified to accept parameters about the target platform and parameters about a high-level language to generate specific code.

11. Conclusion

As Web Services widen in usage, as application topologies continuously evolve to support infrastructures such as firewalls, load balancers, and as awareness of the threats faced by organizations becomes well understood, the need for suitable security specifications for Web Services becomes more explicit. A wide variety of technologies, techniques and implementations are available that address the increasing need of security for distributed applications. However, no solution supports authorization of Web Services in an easy way and no solutions are readily available for deploying simple Web Service based applications. This paper outlines an automatic, efficient, and portable scheme for enforcing authorization policies on Web Services. This approach uses a reusable authorization framework, which saves the developers from tedious and error-prone work, and improves the security of Web Services by decoupling security policies from Web Service specific functionality. The separation of authorization implementation from Web Service application implementation helps in making modification easier and authorization policies maintainable. The features of the framework are best utilized by ARSL, the authorization rule specification language. The main qualifying features of the proposed framework can be summarized as follows: the framework provides a way to decouple authorization implementation from the Web Service application development. The framework together with the specification language, ARSL, constitutes a simple solution for authorization implementation. Since ARSL is based on predicate logic, security administrators can easily master it. It is simple, yet flexible and complete enough to express any complex rule. A banking application was described that served as

a prototype for the proposed authorization framework. The application illustrates the applicability of the authorization framework and the usage of the specification language. The design of the framework makes it suitable to be used for dynamic authorization rule updates. The time taken to generate code in high-level language was small enough not to be a major overhead. In future, this framework can be adapted to other models of authorization and extended to other high-level languages as well.

Works Cited

Adam, Barth, John C. Mitchell, and Justin Rosenstein. "Conflict and combination in privacy policy languages ." 03 Apr 2007. *Proceedings of the 2004 ACM workshop on Privacy in the Electronic Society* 2004 45-46.

Ahn, Gail-Joon, and Ravi Sandhu. "Role-Based Authorization Constraints Specification." *ACM Transactions on Information and System Security (TISSEC)* 3(2000): 207-226.

Alotaiby, Fahad T, and Jim X Chen. "A Model for Team-based Access Control (TMAC 2004)." *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)* 1(2004): 450-454.

Anderson, Anne H. "A Comparison of Two Privacy Policy Languages: EPAL and XACML." 03 APR 2007. *Proceedings of the 3rd ACM workshop on Secure web services* (2006):53-60.

Bertino, Elisa, Claudio Bettini , and Pierangela Samarati. "A discretionary access control model with temporal authorizations." *Proceedings of the 1994 workshop on New security paradigms* (1994): 102-107.

Bertino, Elisa, Claudio Bettini , Elena Ferrari , and Pierangela Samarati. "A Temporal Access Control Mechanism for Database Systems." *IEEE Transactions on Knowledge and Data Engineering* 8 (1996): 67-80.

Bhatti, Rafae, Elisa Bertino, and Arif Ghafoor. "A Trust-Based Context-Aware Access Control Model for Web-Services." *Proceedings of the IEEE International Conference on Web Services (ICWS'04)* (2004): 184-191.

Cawsey, Alison. "Artificial Intelligence." *Databases and Artificial Intelligence 3 Artificial Intelligence Segment.* School of Mathematical and Computer Sciences Heriot-Watt University. 26 Apr. 2005 <http://www.macs.hw.ac.uk/~alison/ ai3notes/ section2_4_3.html>.

Chaari, Sodki, Frederique Biennier, Chokri Ben Amar, and Joel Favrel. "An Authorization and Access Control Model for Workflow." *First International Symposium on Control, Communications and Signal Processing* (2004): 141-148.

Coetzee, M., and J. H. P Eloff. "A Logic-Based Access Control Approach for Web Services." *Proceeding of the 2004 ISSA Information Security Conference* (2004): 1-11.

"Enterprise Privacy Authorization Language (EPAL)." IBM. Vers. 1.2, 2003 <http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/>.

Godik, Simon., and Tim Moses, eds.Extensible Access Control Markup Language". Vers. 1.1. Aug. 2003. 15 Apr. 2005. <http://www.oasis-open.org/committees/xacml /repository/cs-xacml-specification-1.1.pdf>.

Indrakanti, Sarath, Vijay Varadharajan, and Michael Hitchens."Authorization Service for Web Services and its Implementation." *Proceedings of the IEEE International Conference on Web Services (ICWS'04)* (2004): 774-777.

Jajodia, Sushil, Pierangela Samarati, and V.S. Subrahmanian. "A logical language for expressing authorizations." *Proceedings of the Symposium on Security and Privacy* (1997): 31-42.

Levin, John R., Tony Mason, and Doug Brown. *Lex and Yacc.* 2nd ed. O'Reilly &Associates, Inc., 1992.

Mahmoud, Qusay H.. "Securing Web Services." *Java Technologies.* Mar. 2005. 17 Sep. 2005 <http://java.sun.com/developer/technicalArticles/WebServices/ security/>.

Meier, J. D., Alex Mackman, Michael Dunner, and Srinath Vasireddy. "ASP.NET Security." *Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication.* Nov. 2002. 19 Apr. 2005.<http://msdn.microsoft.com/ library/default.asp?url=/library/en-us/dnnetsec/html/SecNetch08.asp>.

Moorthy, Kumar Raj and R.K.Gandhirajan. "The Foundations of Web Services Security." Developer.com. 31 May 2005. <http://www.developer.com/services/article.php/ 3496326>.

Narayanan, Srividhya and Chris Webster. "Securing J2EE Applications in Sun Java Studio Enterprise 8." *Sun Java Studio Enterprise Technical Articles* 18 Oct. 2005. 25 Apr. 2005. <http://developers.sun.com/prodtech/javatools/jsenterprise/reference /techart/jse8/security.html>.

"Network Security Technologies and Protocols: AAA, VPN and Firewall." *Javvin | Network Protocols Guide, Network Monitoring & Analysis Tools.* 17 May 2005

Sandhu, Ravi, and Pierangela Samarati . "Authentication, access control, and audit." *ACM Computing Surveys (CSUR)* 28(1996): 241-243.

Schaad, Andreas, Jonathan Moffett, and Jeremy Jacob. "The Role-Based Access Control System of a European Bank: A Case Study and Discussion." *Proceedings of the sixth ACM symposium on Access control models and Technologies* (2001): 3-9.

Saikali, Ed. "Securing Web Services." Oracle Magazine 17 Jul. 2002. 25 Apr. 2005

<http://www.oracle.com/technology/oramag/oracle/02-jul/o42special_web.html>.

"Security Assertion Markup Language (SAML)." 2004. 12 Mar. 2005 <http://xml.coverpages.org/saml.html>.

Security in the Microsoft ® .NET Framework. 15 Apr. 2005. <http://www.coresecurity.com/files/files/12/dotnet-security-framework.pdf>.

Siddiqui, Bilal . "Web Services Security, Part 1." 04 Mar. 2003. 3 May 2005 <http://webservices.xml.com/pub/a/ws/2003/03/04/security.html>.

Siddiqui, Bilal . "Web Services Security, Part 2." 01 Apr. 2003. 3 May 2005 <http://www.xml.com/pub/a/ws/2003/04/01/security.html?page=1>.

Simon, Ed, Paul Madsen, and Carlisle Adams. "An Introduction to XML Digital Signatures." 08 Aug. 2001. 17 May 2005 <http://www.xml.com/ pub/a/2001/08/08/ xmldsig.html>.

Sirer, Emin Gün, and Ke Wang. "An access control language for web services." *Proceedings of the seventh ACM symposium on Access control models and technologies* (2002): 23-30.

Skalka, Christian, and X Sean Wang. "Trust But Verify: Authorization for Web Services." *Proceedings of the 2004 workshop on Secure web service* (2004): 47-55.

"Verisign Web Services Security Assessment." VeriSign.com. 19 Nov. 2005. <http:// www.verisign.com/stellent/groups/public/documents/white_paper/006970.pdf>.

"Web Services Architecture Requirements." W3C. 14 Nov. 2002. 20 Feb. 2005 <http:// www.w3.org/TR/2002/WD-wsa-reqs-20021114>.

"XML-Signature Syntax and Processing." W3C. 12 Feb. 2002. 17 Oct. 2005 <http:// www.w3.org/TR/xmldsig-core/>.

Weaver, Alfred C. "Enforcing distributed data security via Web services." *Proceedings of the 2004 IEEE International Workshop on Factory Communication Systems* (2004): 22-24.

Weaver, Alfred C., S.J. Dwyer, A.M. Snyder III, J. V an Dyke, J. Hu, X. Chen, T. Mulholland, and A. Marshall. " Federated, secure trust networks for distributed healthcare IT services." *Proceedings of the IEEE International International Conference on Industrial Informatics* (2003): 162-169.

Weaver, Alfred. "A Security Architecture for Data Privacy and Security." *10th IEEE*

Conference on Emerging Technologies and Factory Automation 2005. ETFA 2005 1(2005): 673-676.

"Web Services Security." IBM.com. 01 Mar. 2004. 18 Dec. 2005 <http:// www.128.ibm.com/developerworks/ library/specification/ws-secure/>.

Weeks, Stephen. "Understanding Trust Management Systems." *Proceedings of the 2001 IEEE Symposium on Security and Privacy* (2001): 94-106.

"XML and Web Services Security." java.sun.com. 20 Jan. 2007 <http:// java.sun.com/j2ee/1.4/docs/tutorial/doc/Security7.html>.

Yu, Weider D, and Archana Mansukhani. "An Approach in Designing Authorization Layer for Web Services Software Using Expert System." *Proceedings of the 8th International Conference on Computer Science and Informatics* (2005): 316-321.

Yu, Weider D, and Ellora Nayak. "ARSL: A Language for Authorization Rule Specification in Software Security." *Proceedings of the 11th IEEE International Symposium on Computers and Communications (ISCC 2006)* (2006):54-62.

Ziebermayr, Thomas, and Stefan Probst. "Web Service Authorization Framework." *Proceedings of the IEEE International Conference on Web Services (ICWS'04)* (2004): 614-621.

Appendix A: Application Screen Shots

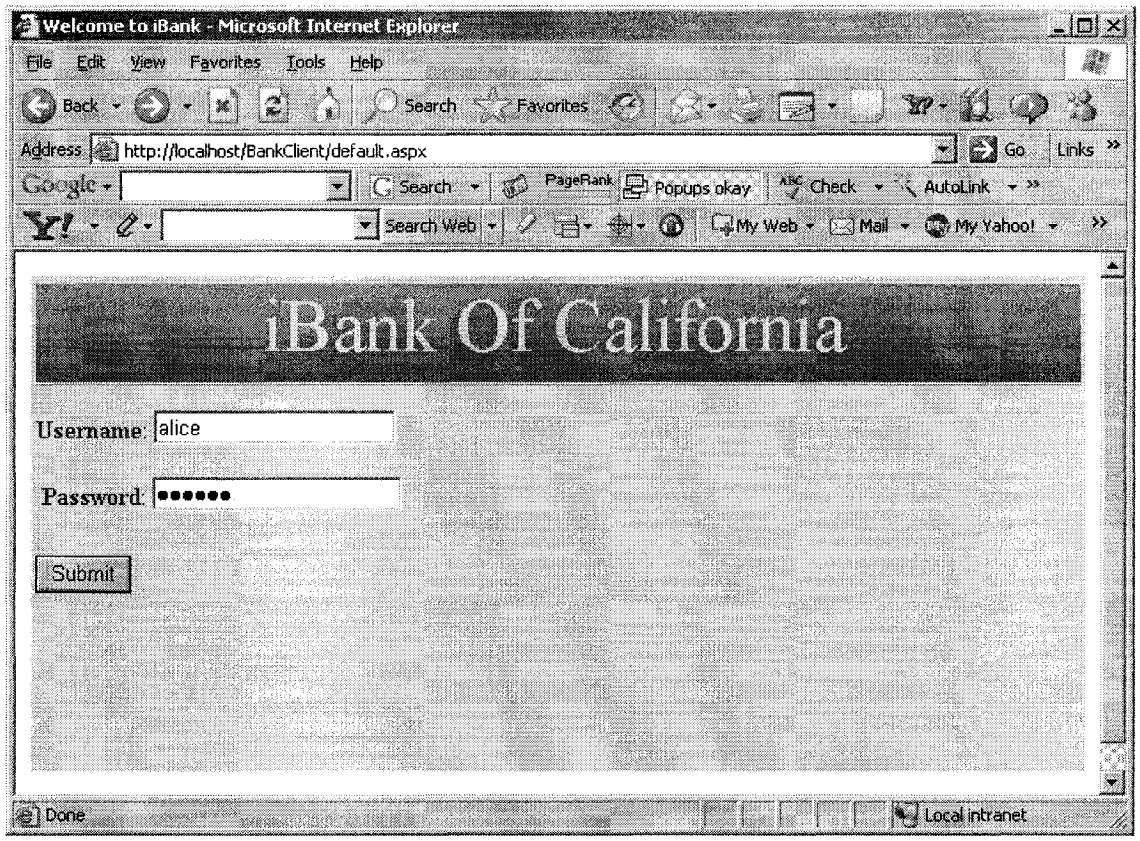


Figure 15: Screenshot 1 – login screen

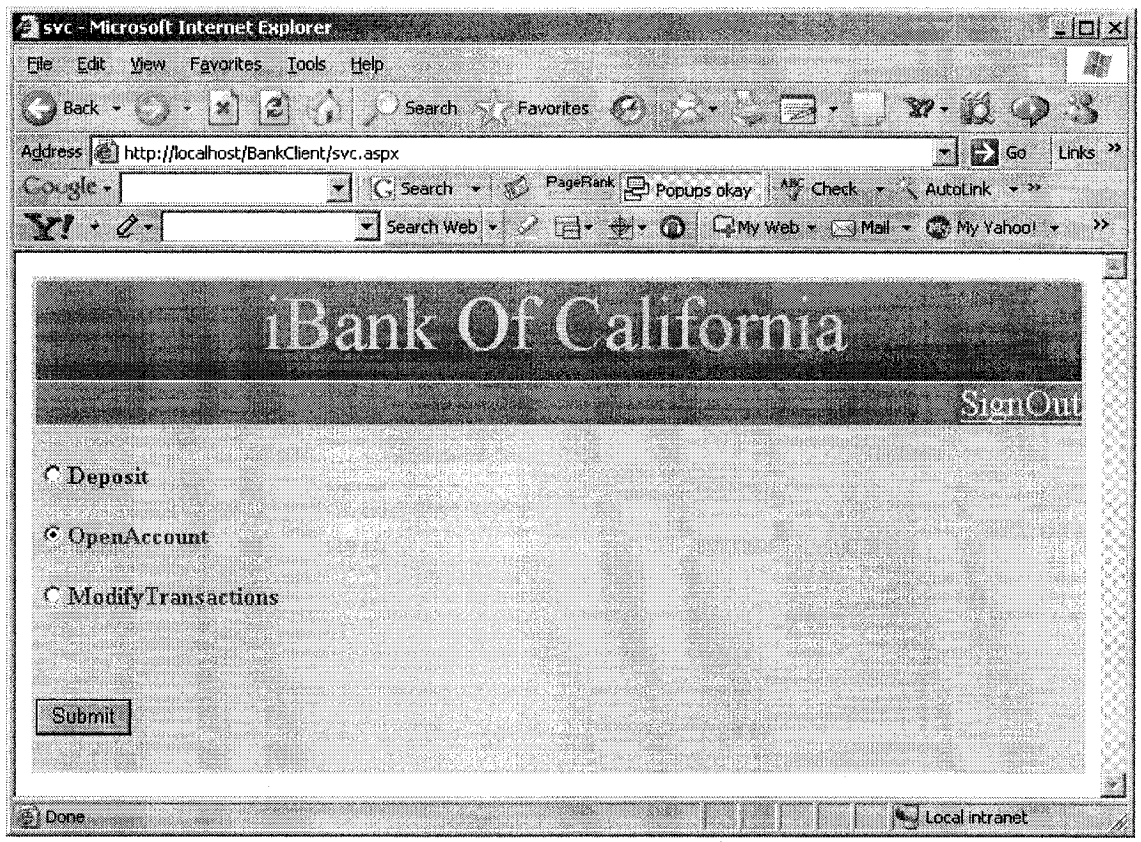


Figure 16: Screenshot 2 – list of available services

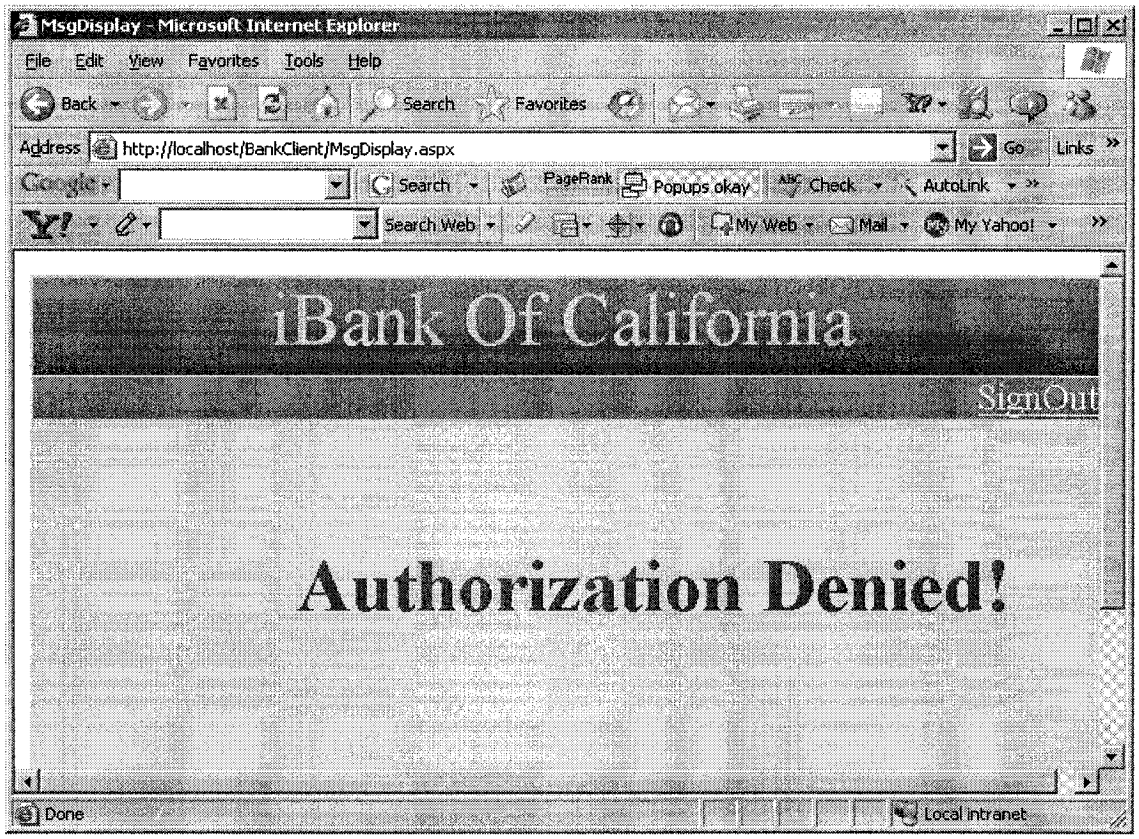


Figure 17: Screenshot 3 – authorization denied screen

Appendix B: Source Code

********** **ARSParser.yxx** **********

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <iostream.h>
#include <fstream.h>
#include <ctype.h>
#include <hash_map.h>
#include <string.h>
#include <slist.h>
#include <hash_set.h>
#include <vector.h>
#include "Services.h"
#include "hoc.tab.hxx"

using namespace std;

#define YYDEBUG 1
#define YYERROR_VERBOSE 1
#define YYPRINT(file, type, value)   yyprint (file, type, value)
#define INDENT "    "

extern int __debug;
int yyerror( const char * s);
string genCode( char * var1, char * op, char * var2);
void appendFuncDecl(  char * cond , char * funcName );
int  appendAsgnStmt( char * var1);
void endEvaluation();
string evaluatePred( char * accessFunc, char * varName);
bool validateResource( char * resource);
string generateMacroFunc( char * clause, char * macroName );
string GeneratePredFuncs(char * funcName, char * arg);
void generate_func();
void genAccesFuncs( string s);
int initialize();
int serviceIdCounter;
#include <FlexLexer.h>
yyFlexLexer* _glob_expr;
vector< vector< int > > accessTypeVector;
Services * serviceMap = Services::CreateServiceMap( "services.txt" );
double _glob_expr_val;
```

```
YYSTYPE *glob_lex_yacc_ptr;
string glob_input;
string codeGenerated;
string  accessFuncStr;
string curRule;
int mycount;
char ctr_buf[5];


slist <int> accessSvcList;

slist <int> denySvcList;
slist <char *> macroDataList;
char * columnName;

struct Node {
   char * funcName;
   struct Node * next;
};

struct Rules {
   string rule;
   string code_gen;
   int access_deny;
   Rules( const string & p_rule, const string & p_code_gen, int p_access_deny
):rule(p_rule), code_gen(p_code_gen), access_deny(p_access_deny)
   {
   }
};

vector< vector< Rules > > rulesVector;

int initialize()
{
   accessTypeVector.reserve(serviceMap->GetServiceMapSize());
   rulesVector.reserve(serviceMap->GetServiceMapSize());
   for (int i = 0; i < serviceMap->GetServiceMapSize(); ++i)
   {
         vector<int> x;
         accessTypeVector.push_back(x);
         vector<Rules> y;
         rulesVector.push_back(y);
   }
}
```

```
bool lookup(const hash_set<const char*, hash<const char*>, eqstr>& Set,
        const char* word)
{
  hash_set<const char*, hash<const char*>, eqstr>::const_iterator it = Set.find(word);
  if( it != Set.end()  ) {
    return true;
  }
    return false;
}
hash_set<const char*, hash<const char*>, eqstr> macroSet;
hash_map<const char*, unsigned int, hash<const char*>, eqstr> accessSvcMap;
hash_map<const char*, unsigned int, hash<const char*>, eqstr> denySvcMap;

%}

%union {
   char * str;
   int num;
   double dbl;
}
%{
int yylex( YYSTYPE * yylval);
void yyprint ( FILE *file, int type, YYSTYPE  value);
%}

%token <num> SERVICE_ID
%token <str> VAR IMPLIES OR AND ASSIGN NOT IF EQUALS BEG END
ACCESS DENY
%token <str> FORALL EXISTS  TRUE FALSE
%token <dbl> NUMBER

%type <str> prog start quantifier  forall exists macro macro_stmt rules
%type <str> access_rule asgn expr  predExpr accessExpr
%left NOT
%left AND
%left GT GE LT LE  NE OR
%left '+' '-'
%left '*' '/'
%pure_parser
%%

start: macro_stmt rules  { $$ = $1; };
rules: BEG prog END  { $$ = $1;
```

```
                    generate_func();
                  serviceMap->generateCsharpFile( codeGenerated );
                  codeGenerated.erase();
                }
    ;

prog: /* nothing */         { $$ = 0  ;  _glob_expr_val = 0;  }
   | prog access_rule
   ;

access_rule: { }
/*    | access_rule asgn */
   | access_rule quantifier '['predExpr IMPLIES accessExpr']' ';'
                  {
                        accessFuncStr = (char*) $4;
/* This is where actual code generation takes place */
                        genAccesFuncs( accessFuncStr );
                       }
;
quantifier: forall exists
          | exists forall
          ;
forall: { }
       | '(' FORALL VAR ')' {
                $$=0;  accessSvcList.clear(); /* List of AccessSvcName()*/
                denySvcList.clear();
 }
          ;
exists:{ }
       | '(' EXISTS VAR ')' { $$=0;  accessSvcList.clear();
                denySvcList.clear();
          }
          ;

asgn:
   VAR ASSIGN VAR         { char * x = strdup(genCode((char *) $1,"=" ,(char *)$3
).c_str());
                        $$ = x;
                  codeGenerated.append(x );
                        }
   | VAR ASSIGN expr       { appendAsgnStmt((char*) $1);
                  codeGenerated.append((char*) $3 );
                        codeGenerated.append( ";\n");
                       }
```

```
  | expr ASSIGN ""VAR""   {
codeGenerated.append(INDENT); codeGenerated.append("char * " );
                              codeGenerated.append((char *) $1);
                              codeGenerated.append(" = " ); codeGenerated.append("\"");
                              codeGenerated.append((char*)$4);
codeGenerated.append("\"");
                        codeGenerated.append( ";\n");
                              }
  ;

accessExpr :  { }
       | accessExpr OR accessExpr { $$ = 0; }
       | accessExpr AND accessExpr {  }
         | ACCESS'(' VAR ')'        {
   hash_map<const char*, unsigned int, hash<const char*>, eqstr>::iterator i;
                        if( validateResource((char *) $3 )) {
                              int serviceId = serviceMap->GetServiceId( ( const char*) $3);
                              accessTypeVector[serviceId].push_back(1);
// accessSvcMap contains list of Services
                     accessSvcList.push_front(serviceId);
// AccessSvcList -list of all funcs AccessSvc_<num>
                        }
                    }
         | DENY'(' VAR ')'        {
        hash_map<const char*, unsigned int, hash<const char*>, eqstr>::iterator i;
                        if( validateResource((char *) $3 )) {
                              int serviceId = serviceMap->GetServiceId( (char*) $3);
                              accessTypeVector[serviceId].push_back(0);
// denySvcMap contains list of Services
                     denySvcList.push_front(serviceId);
                        }
                    }
         ;


predExpr:                       { $$ = 0;}
       | TRUE               {
                                    string s;
                              s.append("true");
                                    curRule.append("true");
                                 $$=strdup(s.c_str());
                                  }
       | FALSE              {
                                    string s;
```

```
                    s.append("false");
                        curRule.append("false");
                    $$=strdup(s.c_str());
                      }
| VAR                 { $$ = (char*)$1;
                      curRule.append("true");
                      }
  | ACCESS'(' VAR ')'   {
                      char buff[80];
                  if( validateResource((char *) $3 )) {
                      strcpy( buff, "Access" );
                      strcat( buff, (char*) $3 );
                      strcat( buff, " ( userId");
                      strcat( buff, ") \n");
                      $$=buff;
                      curRule.append("Access (");
                      curRule.append((char*)$3);
                      curRule.append(")");
                      }
                      }
  | DENY'(' VAR ')'   {
                      char buff[80];
                  if( validateResource((char *) $3 )) {
                      strcpy( buff, "Deny" );
                      strcat( buff, (char*) $3 );
                      strcat( buff, " ( userId");
                      strcat( buff, ") \n");
                      $$=buff;
                      }
                      }
| '(' predExpr ')'  {
                      string s;
                    s.append("(");
                      s.append((char*)$2);
                    s.append(")");
                      $$=strdup(s.c_str());

                      }
| NOT predExpr        {
                      string s;
                    s.append("(");
                    s.append("!");
                      s.append((char*)$2);
                    s.append(")");
```

```
                                $$=strdup(s.c_str());
                              }
    | predExpr OR predExpr  { char * x =
                        strdup(genCode((char *) $1, " || ",(char *) $3 ).c_str());
                                $$ = x;
                              }
    | predExpr AND predExpr { char * x =
                                strdup(genCode((char *) $1, " && " ,(char *) $3 ).c_str());
                                $$ = x;
                              }
    | predExpr EQUALS VAR {
                              string s;
                              s.append( (char*)$1);
                              s.append("==");
                              s.append( (char*)$3);
                              $$ = strdup(s.c_str());
                              }

    | predExpr EQUALS ""VAR"" {
                                    string s = GeneratePredFuncs( (char*)$1,
(char*)$4);
                                    $$ = strdup(s.c_str());
                                }
      | VAR '(' VAR ')'   {
// user should take care whether macro is defined or not.
//                          if( lookup(macroSet, (char*)$1)) {
                    char * x =
                         strdup(evaluatePred( (char*)$1,"userId" ).c_str());
                           $$ = x;
/*                                  } else {
                                        $$=(char*)$1;
                                  } */
                                  curRule.append( "VAR(VAR)");
                              }
    | predExpr GE predExpr  { char * x =
                              strdup(genCode((char *) $1, " >= " ,(char *) $3 ).c_str());
                              $$ = x;
                              }
    | predExpr GT predExpr  { char * x =
                              strdup(genCode((char *) $1, " > " ,(char *) $3 ).c_str());
                              $$ = x;
                              }
    | predExpr LT predExpr { char * x =
                              strdup(genCode((char *) $1, " <= " ,(char *) $3 ).c_str());
```

```
                                $$ = x;
                               }
    | predExpr LE predExpr { char * x =
                                strdup(genCode((char *) $1, " < " ,(char *) $3 ).c_str());
                                $$ = x;
                               }
      | predExpr EQUALS NUMBER {
            char buff[10];
            sprintf(buff, "%f", $3 );
              string str;
                 columnName = (char*) $1;
                 str.append("(usr.");
              str.append(columnName);
              str.append(" == ");
                 str.append(buff);
                 str.append(")");
                 $$= strdup(str.c_str());
                 }

| predExpr GE NUMBER {
            char buff[10];
            sprintf(buff, "%f", $3 );
              string str;
                 columnName = (char*) $1;
                 str.append("(usr.");
              str.append(columnName);
              str.append(" >= ");
                 str.append(buff);
                 str.append(")");
                 $$= strdup(str.c_str());
}
| predExpr NE NUMBER {
            char buff[10];
            sprintf(buff, "%f", $3 );
              string str;
                 columnName = (char*) $1;
                 str.append("(usr.");
              str.append(columnName);
              str.append(" >= ");
                 str.append(buff);
                 str.append(")");
                 $$= strdup(str.c_str());
}
```

```
| predExpr GT NUMBER {
                char buff[10];
                sprintf(buff, "%f", $3 );
                   string str;
                      columnName = (char*) $1;
                      str.append("(usr.");
                   str.append(columnName);
                   str.append(" > ");
                      str.append(buff);
                      str.append(")");
                      $$= strdup(str.c_str());
}

| predExpr LE NUMBER {
                char buff[10];
                sprintf(buff, "%f", $3 );
                   string str;
                      columnName = (char*) $1;
                      str.append("(usr.");
                   str.append(columnName);
                   str.append(" <= ");
                      str.append(buff);
                      str.append(")");
                      $$= strdup(str.c_str());
}
| predExpr LT NUMBER {
                char buff[10];
                sprintf(buff, "%f", $3 );
                   string str;
                      columnName = (char*) $1;
                      str.append("(usr.");
                   str.append(columnName);
                   str.append(" < ");
                      str.append(buff);
                      str.append(")");
                      $$= strdup(str.c_str());
}

     ;

expr:                           { $$ = 0; }
   | VAR                        { $$ = (char*)$1; }
   | '(' expr ')'           { $$ = $2; }
   | NOT expr                   { string s;
```

```
                                s.append("!");
                                s.append((char*)$2);
                                $$=strdup(s.c_str());
                               }
  | expr OR VAR                { char * x =
                                strdup(genCode((char *) $1, " || " , (char *) $3 ).c_str());
                                $$ = x;
                              }
  | expr AND VAR               { char * x =
                                strdup(genCode((char *) $1, " && " , (char *) $3
).c_str());
                                $$ = x;
                               }
  | expr EQUALS VAR                  { char * x =
                                strdup(genCode((char *) $1, " == " , (char *) $3 ).c_str());
                                $$ = x;
                               }
  | expr '+' VAR               { char * x =
                                strdup(genCode((char *) $1, " + " , (char *) $3 ).c_str());
                                $$ = x;
                               }
  | VAR '(' VAR ')'    { char * x =
                         strdup(evaluatePred( (char*)$1,"userId" ).c_str());
                         $$ = x;
                        }
  ;


macro:  VAR '(' VAR ')' {  // For a previously defined macro
                          if( lookup(macroSet, (char*)$1)) {
                            string s;
                            s.append((char*)$1);
                            s.append( "( userId )\n");
//                          s.append((char*)$3); s.append(")");
                            $$= strdup(s.c_str());
                          }

                          }
| VAR'(' VAR ')' EQUALS ""VAR"" {
                    string str;
                    columnName = (char*) $1;
                    str.append("(string.Compare(usr.");
                  str.append(columnName);
                  str.append(".ToString().Trim()");
```

```
                    str.append(",\"");
                    str.append((char*)$7);
                    str.append("\", true) == 0 )"); //{\n");
                    //str.append(INDENT);
                    //str.append("return true;\n");
                    //str.append("}\n");
                    $$= strdup(str.c_str());
                  }
| VAR'(' VAR ')' NE '"'VAR'"' {
                    string str;
                    columnName = (char*) $1;
                    str.append("(string.Compare(usr.");
                  str.append(columnName);
                  str.append(".ToString().Trim()");
                    str.append(",\"");
                    str.append((char*)$7);
                    str.append("\", true) != 0 )"); //{\n");
                    //str.append(INDENT);
                    //str.append("return true;\n");
                    //str.append("}\n");
                    $$= strdup(str.c_str());
                  }
| VAR'(' VAR ')' GE NUMBER {
              char buff[10];
              sprintf(buff, "%f", $6 );
                 string str;
                    columnName = (char*) $1;
                    str.append("(usr.");
                 str.append(columnName);
                 str.append(" >= ");
                    str.append(buff);
                    str.append(")");
                    $$= strdup(str.c_str());
}
| VAR'(' VAR ')' EQUALS NUMBER {
              char buff[10];
              sprintf(buff, "%f", $6 );
                 string str;
                    columnName = (char*) $1;
                    str.append("(usr.");
                 str.append(columnName);
                 str.append(" == ");
                    str.append(buff);
                    str.append(")");
```

```
                $$= strdup(str.c_str());
}
| VAR'(' VAR ')' NE NUMBER {
            char buff[10];
            sprintf(buff, "%f", $6 );
              string str;
                columnName = (char*) $1;
                str.append("(usr.");
              str.append(columnName);
              str.append(" >= ");
                str.append(buff);
                str.append(")");
                $$= strdup(str.c_str());
}

| VAR'(' VAR ')' GT NUMBER {
            char buff[10];
            sprintf(buff, "%f", $6 );
              string str;
                columnName = (char*) $1;
                str.append("(usr.");
              str.append(columnName);
              str.append(" > ");
                str.append(buff);
                str.append(")");
                $$= strdup(str.c_str());
}

| VAR'(' VAR ')' LE NUMBER {
            char buff[10];
            sprintf(buff, "%f", $6 );
              string str;
                columnName = (char*) $1;
                str.append("(usr.");
              str.append(columnName);
              str.append(" <= ");
                str.append(buff);
                str.append(")");
                $$= strdup(str.c_str());
}

| VAR'(' VAR ')' LT NUMBER {
            char buff[10];
            sprintf(buff, "%f", $6 );
```

```
                string str;
                   columnName = (char*) $1;
                   str.append("(usr.");
                str.append(columnName);
                str.append(" < ");
                   str.append(buff);
                   str.append(")");
                   $$= strdup(str.c_str());
}

   | NOT macro                    {
                                   string s;
                              s.append("(");
                              s.append("!");
                                 s.append((char*)$2);
                              s.append(")");
                                    $$=strdup(s.c_str());
                                  }

   | macro OR macro {
char * x=
            strdup(genCode((char *) $1, " || ",(char *) $3 ).c_str());
                 $$ = x;
                    }
   | macro AND macro { char * x =
             strdup(genCode((char *) $1, " && " ,(char *) $3 ).c_str());
                          $$= x;
                        }
   | macro GE macro { char * x =
             strdup(genCode((char *) $1, " >= " ,(char *) $3 ).c_str());
                          $$= x;
                        }
   | '(' macro ')'  {
                          string s;
                          s.append("(");
                          s.append((char*)$2);
                          s.append(")");
                          $$=strdup(s.c_str());
                     }
;


macro_stmt: {}
| macro_stmt macro IMPLIES VAR'(' VAR ')' ';' {
```

```
                    macroSet.insert(strdup((char*)$4)); // insert new macro
                    string s = generateMacroFunc((char*)$2, (char*)$4);
                codeGenerated.append(s);
                    }
;
%%
string GeneratePredFuncs(char * funcName, char * arg)
{
string str;
str.append("string.Compare(");
str.append("usr.");
str.append(funcName);
str.append(".ToString().Trim(),\"");
str.append(arg);
str.append("\", true) == 0 ");
return str;
}


string generateMacroFunc( char * clauses, char * macroName )
{
string str;

//if( lookup(macroSet, *)$1)) {
str.append("\npublic bool  ");
str.append(macroName);
str.append( "( int userId )\n");
str.append( "{\n");
#if 0
str.append("string myVal = usr.");
str.append(columnName);
str.append(".ToString().Trim();\n");
slist<char *>::const_iterator end = macroDataList.end();
slist<char *>::iterator curr;
for( curr = macroDataList.begin(); curr != end; curr++ ) {
        str.append("if(string.Compare(myVal,\"");
        str.append(*curr);
        str.append("\", true) == 0 )\n{\n");
        str.append(INDENT);
        str.append("return true;\n");
        str.append("}\n");
}
#endif
str.append(INDENT);
```

```
str.append("return( ");
str.append(clauses);
str.append(");\n");
//str.append("return false;\n");
str.append("}\n\n");
macroDataList.clear();
return str;
}

void genAccesFuncs( string s)
{
   char * func;
   const slist<int>::iterator end = accessSvcList.end();
   slist<int>::iterator curr;

   for( curr = accessSvcList.begin(); curr != end; curr++ )
   {
      int serviceId = * curr;
         rulesVector[serviceId].push_back( Rules( glob_input, s, 1 ));

   }
   const slist<int>::iterator end1 = denySvcList.end();
   for( curr = denySvcList.begin(); curr != end1; curr++ )
   {
      int serviceId = * curr;
         rulesVector[serviceId].push_back( Rules( glob_input, s, 0 ));
   }
   glob_input.clear();
}

void generate_func( )
{
   int i, j;
   int deny_cnt;
   int access_cnt;
   hash_map<const char*, unsigned int, hash<const char*>, eqstr>::iterator svcMapIter ;
   hash_map<const char*, unsigned int, hash<const char*>, eqstr>::iterator access_iter ;
   hash_map<const char*, unsigned int, hash<const char*>, eqstr>::iterator deny_iter ;
   string tmpStr1, tmpStr2, tmpStr3;
   char *  svcName;
   char tmp[5];
   int numOfSvcs = serviceMap->GetServiceMapSize();
if ( __debug == 0 ) {
   for ( j=1; j < numOfSvcs ; j++ )
```

```
  {
    svcName = serviceMap->GetServiceName(j);
    int serviceId = serviceMap->GetServiceId( svcName);
      int numRules = rulesVector[serviceId].size();
      if ( numRules == 0 )
      {
        continue;
      }
      bool curAction = ( rulesVector[serviceId][numRules -1].access_deny == 1);
      string allRules;
      string curPredicate =  string( "( " ) + rulesVector[serviceId][numRules -
1].code_gen + " )";
      allRules = rulesVector[serviceId][numRules - 1].rule ;
      for ( int k = numRules - 2; k >= 0; --k )
      {
      allRules = rulesVector[serviceId][k].rule + "\n" + allRules;
      bool newAction = rulesVector[serviceId][k].access_deny;
        string newPredicate = string( "( " ) + rulesVector[serviceId][k].code_gen + " )";
        if ( rulesVector[serviceId][k].access_deny == curAction )
        {
            curPredicate = string( "( ") + newPredicate + string ( "\n ") + INDENT +
INDENT + INDENT + string( "|| " ) + curPredicate + string (" )");
        }
        else
        {
            curPredicate = string( "( ! ") + newPredicate + string ( "\n ") + INDENT +
INDENT+ INDENT + string("&& " ) + curPredicate + string (" )");
        }
      }
      if ( !curAction )
      {
        curPredicate = string( " ( ! " ) + curPredicate + string( ")");
      }
      tmpStr1.append ( "\n");
//      tmpStr1.append("[Conditional(\"AVOID_CONFLICT\")]\n");
      tmpStr1.append ( "public bool ");
      tmpStr1.append(" Access");
      tmpStr1.append( svcName);
      tmpStr1.append( "( int userId )\n{\n" );
      tmpStr1.append("/*\n");
      tmpStr1.append( allRules );
      tmpStr1.append("\n*/\n");
      tmpStr1.append(INDENT);
      tmpStr1.append( "return " );
```

```
        tmpStr1.append( curPredicate );
        tmpStr1.append( "; \n}\n " );
    }
    codeGenerated.append( tmpStr1 );
}

/* Generate Conflict Debug Code */
    if ( __debug == 1 )
    {
        for ( j=1; j < numOfSvcs ; j++ )
        {
            svcName = serviceMap->GetServiceName(j);
            int serviceId = serviceMap->GetServiceId( svcName);
            int numRules = rulesVector[serviceId].size();
            if ( numRules == 0 )
            {
                continue;
            }
            string allRules;
            tmpStr2 += string("bool predicateResult = false;\n")+ ("    bool isAccess =
false;\n");
            tmpStr2.append("    bool isDeny = false;\n");
    //      tmpStr2.append("    string rules;\n");
            tmpStr2.append("    StringBuilder rules = new StringBuilder();\n");
            tmpStr2.append("    int firstRule = -1;\n");
    //      allRules = rulesVector[serviceId][numRules - 1].rule ;
            for ( int k = 0; k < numRules; ++k )
            {
            string curRule;
            allRules = rulesVector[serviceId][k].rule + "\n" + allRules;
            string curPredicate =  string( "( " ) + rulesVector[serviceId][k].code_gen +
string (" );\n");
            tmpStr2 += string(INDENT) + ("predicateResult = ");
            tmpStr2.append(curPredicate );
            //check if curPred is Access or Deny
            tmpStr2 += string(INDENT) + ("if ( predicateResult )\n    {\n");
            tmpStr2 += string(INDENT) + INDENT + ("rules.Append(\"");
            curRule = rulesVector[serviceId][k].rule ;

            tmpStr2.append(curRule);
            tmpStr2.append("\\n\");\n");

            if ( rulesVector[serviceId][k].access_deny == 0 )
            {
```

```
                tmpStr2 += string(INDENT) + INDENT + string("isDeny = true;\n");
                tmpStr2 += string(INDENT) + INDENT + string( "if ( firstRule < 0 )\n");
                tmpStr2 += string(INDENT) + INDENT + INDENT + ("{\n");
                tmpStr2 += string(INDENT) + INDENT + INDENT + ("firstRule = 0;
\n") ;
                tmpStr2 += string(INDENT) + INDENT + INDENT + ("}\n");
            }
            if ( rulesVector[serviceId][k].access_deny == 1 )
            {
                tmpStr2 += string(INDENT) + INDENT + string("isAccess = true;\n");
                tmpStr2 += string(INDENT) + INDENT + string( "if ( firstRule < 0 )\n");

                tmpStr2 += string(INDENT) + INDENT + INDENT + ("{\n");
                tmpStr2 += string(INDENT) + INDENT + INDENT + ("firstRule = 1;
\n") ;
                tmpStr2 += string(INDENT) + INDENT + INDENT + ("}\n");
            }
            tmpStr2 += string(INDENT) + ("}\n");
        }
        tmpStr2 += string(INDENT) +("if( isAccess && isDeny ) \n") + INDENT
+("{\n");
        tmpStr2 += string(INDENT) + INDENT + ("StreamWriter w =
File.AppendText(FILE_NAME);\n");
        tmpStr2 += string(INDENT) +  INDENT +("w.WriteLine(\"The following rules
have evaluated to true for user {0} resulting in conflict.\n\", usr.FirstName);\n");
        tmpStr2 += string(INDENT) + INDENT + ("string tmp = \"Resolving to \" +
(firstRule == 1 ? \"Access\" : \"Deny\") ;\n") +
INDENT+INDENT+("rules.Append(tmp);\n");
        tmpStr2 += string(INDENT) + INDENT + ("w.WriteLine(rules);\n") + INDENT
+ INDENT + ("w.Close();\n") + INDENT + INDENT + ("return ( firstRule == 1 ? true :
false );\n");
        tmpStr2 += string(INDENT) + ("} \n") + INDENT + ("else if( isAccess || isDeny
)\n") + INDENT + ("{\n")+ INDENT+ INDENT +("return (isAccess);\n") + INDENT +
("}\n");

    tmpStr2 += string(INDENT)+ ("return true; /* Based on the default rule. */ \n");


        tmpStr3.append ( "\n");
    //  tmpStr3.append("[Conditional(\"DEBUG_CONFLICT\")]\n");
        tmpStr3.append ( "public bool ");
        tmpStr3.append(" Access");
        tmpStr3.append( svcName);
        tmpStr3.append( "( int userId )\n{\n" );
```

```
        tmpStr3.append("/*\n");
        tmpStr3.append( allRules );
        tmpStr3.append("\n*/\n");
        tmpStr3.append(INDENT);
        tmpStr3.append( tmpStr2 );
        tmpStr3.append( " \n}\n " );
        }
        codeGenerated.append( tmpStr3 );
    }
}

int  appendAsgnStmt( char * var1)
{
   string tmpStr;
   codeGenerated.append(INDENT);
   codeGenerated.append( "bool " );
   codeGenerated.append(var1);
   codeGenerated.append( " = " );
   return tmpStr.size();
}

void appendFuncDecl(  char * cond , char * funcName )
{
   string tmpStr, strEnd;
   tmpStr.append("\npublic bool  ");
   tmpStr.append( funcName );
   tmpStr.append( "( int userId ) { \n" );
   codeGenerated.insert( 0, tmpStr );
   strEnd.append(INDENT);
   strEnd.append( "if ( ");
   strEnd.append( cond );
   strEnd.append( " )  { \n");
   strEnd.append(INDENT);
   strEnd.append(INDENT);
   strEnd.append("return true ; \n");
   strEnd.append(INDENT);
   strEnd.append( "} \n}" );
   codeGenerated.append( strEnd );

}
char *progname;

string evaluatePred( char * accessFunc, char * varName)
{
```

```
  string tmpStr;
  tmpStr.append( accessFunc );
  tmpStr.append( "( ");
  tmpStr.append(varName);
  tmpStr.append( " )\n");
  return tmpStr;
}

string genCode( char * var1, char *  op, char * var2)
{
   string tmpStr;
   if (strcmp( op, "=" ) == 0 ) {
      tmpStr.append(INDENT);
         tmpStr.append( "\npublic bool  ");
         tmpStr.append( var1);
         tmpStr.append( " = query( userId,");
         tmpStr.append( var2 );
         tmpStr.append( ");\n");
   } else {
         // tmpStr.append( "( " );
         tmpStr.append( var1 );
         tmpStr.append( op );
         tmpStr.append( var2 );
         //tmpStr.append( " )\n" );
   }
#if 0
   serviceMap->UpdateFunctionCode( codeGenerated, serviceId );
#endif
   return tmpStr;
}
bool validateResource( char * resource)
{
   unsigned int serviceId = serviceMap->GetServiceId( resource );
   if ( serviceId ) {
         return true;
   } else {
         cerr << "Invalid Service " << resource << endl;
         exit(-1);
   }

}


double evalExprFile( const char *fname )
```

```
{
   _glob_expr = new yyFlexLexer(new ifstream( fname ));
   initialize();
   yyparse();
}

void endEvaluation()
{
   Services::DestroyServiceMap( serviceMap );
}

int yylex(YYSTYPE *ptr)
{
   glob_lex_yacc_ptr = ptr;
   return _glob_expr->yylex();
}

void yyprint ( FILE *file, int type, YYSTYPE value)
{
 if (type == VAR)
   fprintf (file, "%s", value.str);
 else if (type == NUMBER)
   fprintf (file, "%f", value.dbl);
}

#if 1
int yyerror( const char * s )
{
    printf("%s at line %d with \"%s\"\n", s, _glob_expr->lineno(), s);
  // fprintf(stderr, "%s  \n",  s);
    exit (-1);
   //  return 0;
}
#endif
```

```
**********************        ARSLexer.l        ************************
%{
#include "hoc.tab.hxx"
#include <stdlib.h>
#include <iostream.h>
#include <string.h>
#include <string>
using std::string;


extern YYSTYPE *glob_lex_yacc_ptr;
extern string glob_input;
#define LOOKUP 0
char * predicate;
int addWord( int type, char * word );
int lookUp(char * word ) ;
%}

%%
[ \t]   { glob_input += yytext; }
[0-9]+  { glob_lex_yacc_ptr->dbl = atoi(yytext); glob_input += yytext; return NUMBER;
}

OR      { glob_input += yytext; return OR; }
AND     { glob_input += yytext; return AND ;}
NOT     { glob_input += yytext; return NOT ; }
EQUALS   { glob_input += yytext; return EQUALS; }
ASSIGN   { glob_input += yytext; return ASSIGN; }
BEGIN    { return BEG; }
END      { return END; }

"=>"     { glob_input += yytext; return IMPLIES; }
"||"    { glob_input += yytext; return OR; }
">="     { glob_input += yytext; return GE; }
">"      { glob_input += yytext; return GT; }
"<="     { glob_input += yytext; return LE; }
"<"      { glob_input += yytext; return LT; }
"&&"     { glob_input += yytext; return AND; }
"!"      { glob_input += yytext; return NOT; }
"!="     { glob_input += yytext; return NE; }
"=="     { glob_input += yytext; return EQUALS; }
"="      { glob_input += yytext; return ASSIGN; }
"if"     { glob_input += yytext; return IF; }
```

```
"forall"   { glob_input += yytext; return FORALL; }
"exists"   { glob_input += yytext; return EXISTS; }
"Access"   { glob_input += yytext; return ACCESS; }
"Deny"   { glob_input += yytext; return DENY; }
"TRUE"  { glob_input += yytext; return TRUE; }
"FALSE"  { glob_input += yytext; return FALSE; }


[a-zA-Z0-9_]+ { glob_input += yytext; glob_lex_yacc_ptr->str = strdup( yytext); return
VAR; }
\n        { yylineno++; }
.         {
            if (yytext[0] == "") {
                  glob_input += "\\";
            }
            glob_input += yytext;
            return yytext[0];
          }


%%


****************************** Services.cxx ******************************

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include "Services.h"

Services::Services( FILE * fptr )
{
   genFile.open("GeneratedFile.cs", ios::out);
   char buff[1024];
   int cnt = 0;
   char *duplicate = strdup( "INVALID" );
   services.push_back( duplicate );
   nameToIdMap[duplicate] = cnt++;

   for ( ; fscanf( fptr, "%s\n", buff ) != EOF ; ) {
         duplicate =  strdup(buff) ;
         services.push_back( duplicate );
         nameToIdMap[duplicate] = cnt++;
   }
```

```
    codeArray = new string[services.size()];
}

Services::~Services()
{
    delete [] codeArray;
    nameToIdMap.clear();
    for( int i = 0; i < services.size(); i++ ) {
        free ( services[i] );
    }
    services.clear();
    genFile.close();
}

Services * Services::CreateServiceMap( const char * filename )
{
    FILE * fptr = fopen ( filename, "r" );
    if ( !fptr ) {
        return NULL;
    }

    return (new Services(fptr));
}

void Services::DestroyServiceMap( Services *ptr )
{
    delete ptr;
}

unsigned int Services::GetServiceId( const char * serviceName ) const
{
    hash_map<const char*, unsigned int, hash<const char*>, eqstr>::const_iterator const i
= nameToIdMap.find(serviceName);

    if ( i == nameToIdMap.end() ) {
        return 0;
    }
    return i->second;
};

unsigned int Services::GetServiceMapSize() const
{
    return services.size();
```

```
}
char * Services::GetServiceName( unsigned int serviceId ) const
{
   if ( serviceId >= services.size() ) {
         return NULL;
   }
   return services[(size_t)serviceId];
}

void Services::UpdateFunctionCode( const string & s, int serviceId )
{
   codeArray[serviceId].append( s );
}

void Services::printCode( int id ) const
{
   cout << codeArray[id] << endl;
}

void Services::generateCsharpFile( const string &  code )
{
   if (genFile)
   {
         genFile << code;
   }
}

**********Services.h**********
#define MAX_SERVICE 100

#include <stdio.h>
#include <hash_map.h>
#include <vector.h>
#include <string>

using namespace std;


struct eqstr
{
 bool operator()(const char* s1, const char* s2) const
 {
  return strcmp(s1, s2) == 0;
 }
```

```
};


class Services
{
   vector<char *> services;
   string * codeArray;
   hash_map<const char*, unsigned int, hash<const char*>, eqstr> nameToIdMap;
   ofstream genFile;

   Services(FILE *fptr);
 public :
   unsigned int GetServiceId( const char * serviceName ) const;
   unsigned int GetServiceMapSize() const;
   char * GetServiceName( unsigned int serviceId ) const;

   static Services * CreateServiceMap( const char * fileName );
   static void DestroyServiceMap( Services *ptr );

   void generateCsharpFile( const string & code );
   void printCode( int id ) const;
   void UpdateFunctionCode( const string & s, int serviceId );



   ~Services();
};
```

********** compile.sh **********

```
#!/bin/sh
#compile input using our compiler to generate authorization code
if [ $# -ne 1 ]
then

        echo "Usage: ./compile.sh <input_file>"
          exit 0
 fi

#make clean
make
LOGDIR=`pwd`
LOGFILE=$LOGDIR/logs_`date +%m%d%y`
DIR=C:/299/Project/AuthLib/AuthLib
BACKUPDIR=C:/299/Project/299ProjBackUp/AuthLib



echo "Compiling  $1 ..."
echo "Compiling  $1 ..." >> $LOGFILE
echo "Step 1. ars $1."
echo "Step 1. ars $1." >> $LOGFILE
# Using the –D compiler flag
./ars -D $1

echo " " >> $LOGFILE
echo " " >> $LOGFILE
echo " " >> $LOGFILE
echo " "
echo "Step 2. "
echo "replace.pl $BACKUPDIR/Authorize.cs $DIR GeneratedFile.cs".
echo "Step 2. " >> $LOGFILE
echo "replace.pl $BACKUPDIR/Authorize.cs $DIR GeneratedFile.cs". >> $LOGFILE
./replace.pl $BACKUPDIR/Authorize.cs $DIR GeneratedFile.cs
echo "Inserted GeneratedFile.cs within Authorize.cs."
echo "Inserted GeneratedFile.cs within Authorize.cs." >> $LOGFILE

echo " "
echo " " >> $LOGFILE
echo " " >> $LOGFILE
echo " " >> $LOGFILE
```

```
echo "Step 3. COMPILATION OF AUTHORIZATION MODULE" >> $LOGFILE
echo " *****  BEGIN OF COMPILATION OF AUTHORIZATION MODULE *****"
>> $LOGFILE
echo "Step 3. COMPILATION OF AUTHORIZATION MODULE"
# generate Authorization dll
$DIR/compileAuth.bat >> $LOGFILE
echo "***** END OF  COMPILATION OF AUTHORIZATION MODULE *****" >>
$LOGFILE
echo " " >> $LOGFILE
echo " " >> $LOGFILE
echo " " >> $LOGFILE
echo " "
# compile & link web service
WSDIR=C:/Inetpub/wwwroot/BOC
echo "Step 4. COMPILATION OF WEB SERVICE MODULE "
echo "Step 4. COMPILATION OF WEB SERVICE MODULE " >> $LOGFILE
echo " "
echo "***** BEGIN OF COMPILATION OF WEB SERVICE MODULE *****" >>
$LOGFILE
$WSDIR/compileBOC.bat >> $LOGFILE
echo "*****  END OF COMPILATION OF WEB SERVICE MODULE *****" >>
$LOGFILE
echo " "
echo "Check Log File  $LOGFILE  for more details."

echo " "
#CLNT_DIR=C:/Inetpub/wwwroot/BankClient
#$CLNT_DIR/compileClient.bat
```

```
************************** SampleInput File **************************
Role(x)=="TLR" => Role_TLR(x);
Role(x)=="CSR" => Role_CSR(x);
Role(x)=="LTR" => Role_LTR(x);
Role(x)=="ACC" => Role_ACC(x);
Role(x)=="ASM" => Role_ASM(x);
Role(x)=="BRM" => Role_BRM(x);
Role(x)=="PBK" => Role_PBK(x);
Role(x)=="SVM" => Role_SVM(x);

BEGIN

(forall x) [ Role_BRM(x) OR Role_ACC(x) OR  NOT AfterHours(x) =>
Access(AFTER_HR_ACCESS)];
(forall x) [ true AND NOT Role_TLR(x) => Access(EDIT_ACCT_INFO)];
(forall x) [ Role_LTR(x) OR Role_SVM(x) => Access( EDIT_TRXN)];

(forall x) [ Role_PBK(x) OR NOT Role_TLR(x) => Access(OPEN_ACT)];
(forall x) [ true AND NOT Role_CSR(x) => Access(DEPOSIT)];

(forall x) [ Role_ASM(x) OR Role_BRM(x) AND NOT (Role_TLR(x) OR Role_LTR(x)
OR Role_CSR(x)) => Access(GEN_LEDG_REPORTS) AND Access(
MODIFY_LEGD)];
(forall x) [ Role_LTR(x)  AND NOT (Role_TLR(x)  OR  Role_ACC(x)) =>
Access(CRT_LOAN_ACCT)  AND   Access( MOD_LOAN_ACCT)];

END
```

```
******************** Sample Input for Conflict Resolution ******************
Role(x)=="TLR" => Role_TLR(x);
Role(x)== 13 => Role_BRM(x);
Role(x)=="EMP" OR Role_TLR(x) OR Role_BRM(x) => Role_EMP(x);

BEGIN
(forall x )[ROLE_TLR(x) => Access(OPEN_ACCT) AND Access(Item)];
(forall x )[ROLE_EMP(x) AND AAA == 12 => Deny(OPEN_ACCT)];
(forall x )[TRUE => Access(OPEN_ACCT)];
END
```

********** Generated Authorization Code**********

With Conflict Detection:

```
public bool  AccessItem( int userId )
{
/*
(forall x)[TRUE => Access(Item) ];
(forall x)[Country =="GERMANY" && Age < 21  && Item == "Liquor" =>
Deny(Item)];
(forall x)[Country =="USA" && Age < 19  && Item == "Liquor" =>
Deny(Item)];
(forall x)[Prescription == "Item" => Access(Item)];
(forall x)[CreditCard=="INVALID" => Deny(Item)];

*/
   bool predicateResult = false;
   bool isAccess = false;
   bool isDeny = false;
   StringBuilder rules = new StringBuilder();
   int firstRule = -1;
   predicateResult = (
string.Compare(usr.CreditCard.ToString().Trim(),"INVALID", true) == 0  );
   if ( predicateResult )
   {
      rules.Append("(forall x)[CreditCard=="INVALID" => Deny(Item)];\n");
      isDeny = true;
      if ( firstRule < 0 )
         {
         firstRule = 0;
         }
   }
   predicateResult = ( string.Compare(usr.Prescription.ToString().Trim(),"Item",
true) == 0  );
   if ( predicateResult )
   {
      rules.Append("(forall x)[Prescription == "Item" => Access(Item)];\n");
      isAccess = true;
      if ( firstRule < 0 )
         {
         firstRule = 1;
         }
   }
```

```
   predicateResult = ( string.Compare(usr.Country.ToString().Trim(),"USA", true)
== 0  && (usr.Age < 19.000000) &&
string.Compare(usr.Item.ToString().Trim(),"Liquor", true) == 0  );
   if ( predicateResult )
   {
      rules.Append("(forall x)[Country =="USA" && Age < 19  && Item ==
"Liquor" => Deny(Item)];\n");
      isDeny = true;
      if ( firstRule < 0 )
         {
         firstRule = 0;
         }
   }
   predicateResult = (
string.Compare(usr.Country.ToString().Trim(),"GERMANY", true) == 0  &&
(usr.Age < 21.000000) && string.Compare(usr.Item.ToString().Trim(),"Liquor",
true) == 0  );
   if ( predicateResult )
   {
      rules.Append("(forall x)[Country =="GERMANY" && Age < 21  && Item
== "Liquor" => Deny(Item)];\n");
      isDeny = true;
      if ( firstRule < 0 )
         {
         firstRule = 0;
         }
   }
   predicateResult = ( true );
   if ( predicateResult )
   {
      rules.Append("(forall x)[TRUE => Access(Item) ];\n");
      isAccess = true;
      if ( firstRule < 0 )
         {
         firstRule = 1;
         }
   }
   if( isAccess && isDeny )
   {
      StreamWriter w = File.AppendText(FILE_NAME);
      w.WriteLine("The following rules have evaluated to true for user {0}
resulting in conflict.
", usr.FirstName);
      string tmp = "Resolving to " + (firstRule == 1 ? "Access" : "Deny") ;
```

```
      rules.Append(tmp);
      w.WriteLine(rules);
      w.Close();
      return ( firstRule == 1 ? true : false );
   }
   else if( isAccess || isDeny )
   {
      return (isAccess);
   }
   return true; /* Based on the default rule. */

}
```

With Conflict Prevention:

```
public bool  AccessItem( int userId )
{
/*
(forall x)[CreditCard=="INVALID" => Deny(Item)];
(forall x)[Prescription == "Item" => Access(Item)];
(forall x)[Country =="USA" && Age < 19  && Item == "Liquor" =>
Deny(Item)];
(forall x)[Country =="GERMANY" && Age < 21  && Item == "Liquor" =>
Deny(Item)];
(forall x)[TRUE => Access(Item) ];
*/
   return ( ! ( string.Compare(usr.CreditCard.ToString().Trim(),"INVALID", true)
== 0  )
         && ( ( string.Compare(usr.Prescription.ToString().Trim(),"Item", true)
== 0  )
         || ( ! ( string.Compare(usr.Country.ToString().Trim(),"USA", true) == 0
&& (usr.Age < 19.000000) &&
string.Compare(usr.Item.ToString().Trim(),"Liquor", true) == 0  )
         && ( ! ( string.Compare(usr.Country.ToString().Trim(),"GERMANY",
true) == 0  && (usr.Age < 21.000000) &&
string.Compare(usr.Item.ToString().Trim(),"Liquor", true) == 0  )
         && ( true ) ) ) ) );
}
```

```
************************** Authorize.cs ******************************
using System;using System.IO;
using System.Reflection;
using System.Collections;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

using System.Windows.Forms;



//using System.Windows.Forms;

namespace AuthLib
{
        /// <summary>
        /// Summary description for Class1.
        /// </summary>
        public class Authorize
        {
                private const string ACCOUNT_KEY = "ACCOUNT_KEY"
                public static EmployeeInfo usr;
                public Authorize()
                { }

                 public bool VerifyAccess(string service)
                {
                        Authorize a = new Authorize();
                        StringBuilder accessFunc = new StringBuilder ("Access");
                        accessFunc.Append(service);


                        Type t = typeof(Authorize);
                        object [] args = new object[] {usr.EmpId};
                        bool val = (bool) t.InvokeMember(accessFunc.ToString(),
                                        BindingFlags.InvokeMethod, null, a, args);

                        return val;
                }
```

```
        public bool VerifyLogin(string txtUserId, string txtPasswd, string service)
        {
                // Get the user input
                string userId = CleanString.InputText(txtUserId, 20);
                string password = CleanString.InputText(txtPasswd, 20);
                StringBuilder accessFunc = new StringBuilder ("Access");
                accessFunc.Append(service);

                //  verify userId & password; return User object
                usr =  DBUtil.verifyLoginInfo( userId, password);
                if( usr != null )
                {
                        Authorize a = new Authorize();

                        Type t = typeof(Authorize);
                        object [] args = new object[] {usr.EmpId};
                        bool val = (bool) t.InvokeMember(accessFunc.ToString(),
                                        BindingFlags.InvokeMethod, null, a,
args);
                        return val;
                }
                return false;
        }

        public UserInfo GetUserInfo(bool required)
        {
                UserInfo myAccount =
                      (UserInfo)HttpContext.Current.Session[ACCOUNT_KEY];

                if (myAccount == null)
                {
                        return null;
                }
                else
                {
                        return myAccount;
                }
        }


/****** Generated Code Inserted Here *****/
public bool  Role_TLR( int userId )
{
```

```
return( (string.Compare(usr.Role.ToString().Trim(),"TLR", true) == 0 ));
}

public bool  Role_CSR( int userId )
{
return( (string.Compare(usr.Role.ToString().Trim(),"CSR", true) == 0 ));
}

public bool  Role_LTR( int userId )
{
return( (string.Compare(usr.Role.ToString().Trim(),"LTR", true) == 0 ));
}

public bool  Role_ACC( int userId )
{
return( (string.Compare(usr.Role.ToString().Trim(),"ACC", true) == 0 ));
}

public bool  Role_ASM( int userId )
{
return( (string.Compare(usr.Role.ToString().Trim(),"ASM", true) == 0 ));
}

public bool  Role_BRM( int userId )
{
return( (string.Compare(usr.Role.ToString().Trim(),"BRM", true) == 0 ));
}

public bool  Role_PBK( int userId )
{
return( (string.Compare(usr.Role.ToString().Trim(),"PBK", true) == 0 ));
}

public bool  Role_SVM( int userId )
{
return( (string.Compare(usr.Role.ToString().Trim(),"SVM", true) == 0 ));
}

public bool  AccessAFTER_HR_ACCESS ( int userId )
{
   return ( Role_BRM( userId )
 || Role_ACC( userId )
 || (!AfterHours( userId )
) ) ;
```

```
}
public bool  AccessEDIT_ACCT_INFO ( int userId )
{
   return ( true && (!Role_TLR( userId )
) ) ;
}
public bool  AccessEDIT_TRXN ( int userId )
{
   return ( Role_LTR( userId )
 || Role_SVM(x) ) ;
}
public bool  AccessOPEN_ACT ( int userId )
{
   return ( Role_PBK( userId )
 || (!Role_TLR( userId )
) ) ;
}
public bool  AccessDEPOSIT ( int userId )
{
   return ( true && (!Role_CSR( userId )
) ) ;
}
public bool  AccessMODIFY_LEGD ( int userId )
{
   return ( Role_ASM( userId )
 || Role_BRM( userId )
 && (!(Role_TLR( userId )
 || Role_LTR( userId )
 || Role_CSR( userId )
)) ) ;
}
public bool  AccessGEN_LEDG_REPORTS ( int userId )
{
   return ( Role_ASM( userId )
 || Role_BRM( userId )
 && (!(Role_TLR( userId )
 || Role_LTR( userId )
 || Role_CSR( userId )
)) ) ;
}
public bool  AccessMOD_LOAN_ACCT ( int userId )
{
   return ( Role_LTR( userId )
 && (!(Role_TLR( userId )
```

```
|| Role_ACC( userId )
)) ) ;
}
public bool  AccessCRT_LOAN_ACCT ( int userId )
{
   return ( Role_LTR( userId )
 && (!(Role_TLR( userId )
 || Role_ACC( userId )
)) ) ;
}}
/** End of Inserted Code **/

/** Begin of Fixed part of authorization Module **/
public bool  AfterHours ( int userId )
{
        int hr=20;
        int min=0;
        DateTime currTime = DateTime.Now;
        if ((currTime.Hour >= hr) && (currTime.Minute >= min))
        {
                return true;
        }
        else
        {
                return  false;
        }
}

        public bool  HasOnlineAccess ( int userId )
        {
                return true;
        }
        public bool  PMA_Membership ( int userId )
        {
                return true;
        }


   } // End of Authorize class
} // End of AuthLib Namespace
```

```
********** DBUtil.cs**********
/*This class is part of the authorization module and contains the database access
functions*/
using System;
using System.Data;
using System.Collections;
using System.Configuration;
using Microsoft.ApplicationBlocks.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace AuthLib
{
        /// <summary>
        /// Summary description for DBUtil.  This class deals with all database access
        /// functionalities for the authorization module.
        /// </summary>
        public class DBUtil
        {

                public static readonly string connStr;
                private static SqlConnection conn;

                public DBUtil()
                {}

                static DBUtil()
                {
                        connStr =
ConfigurationSettings.AppSettings["ConnectionString"];
                }

        public static EmployeeInfo verifyLoginInfo( string username, string password )
                {

                        SqlDataReader rdr = null;
                        conn = new SqlConnection(connStr) ;
        DataSet ds = new DataSet();
                        SqlParameter [] roleParams = new SqlParameter[1];
                        roleParams[0] = new SqlParameter("@roleId", SqlDbType.Int);
                        SqlParameter [] loginParams = new SqlParameter[2];

                        loginParams[0] = new
SqlParameter("@username",SqlDbType.NVarChar, 20);
```

```
                loginParams[0].Value = username;
                loginParams[1] = new SqlParameter("@password",
SqlDbType.NVarChar, 10);
                loginParams[1].Value = password;
        try
                {
                        try
                        {
                                conn.Open();
                        }
                        catch
                        {
                                conn.Close();
                                throw;
                        }

                        rdr =  SqlHelper.ExecuteReader(conn,
CommandType.StoredProcedure,"GetEmployee", loginParams);

                        if (rdr.Read())
                                                        {
                                return new EmployeeInfo(rdr.GetInt32(0),
rdr.GetString(1), rdr.GetString(2),rdr.GetString(3), rdr.GetString(4)  ); //,
rdr.GetInt32(10));
                        }
                }
                catch
                {
                        throw;
                }
                finally
                {
                        if(rdr != null)
                                ((IDisposable)rdr).Dispose();

                        if(conn != null)
                                conn.Dispose();
                }
                return null;
        }
    }
}
```

```
************************** EmployeeInfo.cs*****************************
using System;namespace AuthLib{   public class EmployeeInfo    {                  private
int _empId;              private string _firstName;              private string _lastName;
        private string _email;           private string _role;
                public EmployeeInfo()                {}
        public EmployeeInfo( int empId, string firstName, string lastName,
                                string email, string role)                          {
        this._empId = empId;
                        this._email    = email;
                        this._firstName = firstName;                    this._lastName =
lastName;
                        this.Role=role;          }
        public int EmpId
                {
                        get { return _empId; }
                        set { _empId = value; }
                }
                public string FirstName                 {                       get { return
_firstName; }                   set { _firstName = value; }             }
                public string LastName                  {                       get { return
_lastName; }                    set { _lastName = value; }              }
                public string Email
                {
                        get { return _email; }
                        set { _email = value; }
                }
                public string Role
                {
                        get { return _role; }
                        set { _role = value; }
                }       }}

************************** CompileAuthLib.bat ************************

echo ******** COMPILING AUTHORIZATION  MODULE *********
set DIR=C:\299\Project\AuthLib\AuthLib\
set dll_PATH=C:\Program Files\Microsoft Application Blocks for .NET\Data Access
v2\Code\CS\Microsoft.ApplicationBlocks.Data\bin\Debug\
set WIN_dll=C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\


SET CSC_PATH=%SYSTEMROOT%\Microsoft.NET\Framework\v1.1.4322\
```

```
%CSC_PATH%csc.exe /debug  /t:library
/r:%DIR%bin\Debug\Microsoft.ApplicationBlocks.Data.dll
/r:%WIN_dll%System.Windows.Forms.dll /r:%WIN_dll%System.Data.dll
/out:C:\Inetpub\wwwroot\BOC\bin\AuthLib.dll  %DIR%*.cs

echo ***** END OF COMPILATION OF  AUTHORIZATION  MODULE
```

********** Cleanstring.cs**********

```
/*This program ensures that user input doesn't consist of illegal characters*/
using System;
using System.Text;

namespace AuthLib
{
        /// <summary>
        /// Summary description for CleanString.
        /// </summary>
                public sealed class CleanString
                {
                        public static string InputText(string inputString, int maxLength)
                        {
                                StringBuilder retVal = new StringBuilder();
                                // check incoming parameters for null or blank string
                                if ((inputString != null) && (inputString != String.Empty))
                                {
                                        inputString = inputString.Trim();
                                        //chop the string incase the client-side max length
                                        //fields are bypassed to prevent buffer over-runs
                                        if (inputString.Length > maxLength)
                                        inputString = inputString.Substring(0, maxLength);

                                        //convert some harmful symbols incase the regular
                                        //expression validators are changed
                                        for (int i = 0; i < inputString.Length; i++)
                                        {
                                                switch (inputString[i])
                                                {
                                                        case "":
                                                                retVal.Append("&quot;");
                                                                break;
```

```
                                                    case '<':
                                                            retVal.Append("&lt;");
                                                            break;
                                                    case '>':
                                                            retVal.Append("&gt;");
                                                            break;
                                                    default:

        retVal.Append(inputString[i]);
                                                            break;
                                                }
                                            }

                                            // Replace single quotes with white space
                                            retVal.Replace("'", " ");
                                        }
                                        return retVal.ToString();
                                    }
                                }
}
```

Banking WebService Application

********** BankingWS.asmx.cs**********

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml;

using AuthLib;

namespace BOC {
        /// <summary>
        /// Summary description for Service1.
        /// </summary>
        [WebService(Namespace="http://localhost/BOC/")]
        public class BankingService : System.Web.Services.WebService
        {
                public AuthHeaderCS sHeader;
                public Authorize auth ;
                public BankingService()
                {
                        auth = new Authorize();
                        InitializeComponent();
                }

                #region Component Designer generated code

                //Required by the Web Services Designer
                private IContainer components = null;

                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                private void InitializeComponent()
                {
                }
```

```
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
                if(disposing && components != null)
                {
                        components.Dispose();
                }
                base.Dispose(disposing);
        }

        #endregion

        [WebMethod(Description="This method requires a custom soap header set
by the caller")]
        [SoapHeader("sHeader")]
        public EmployeeInfo SignIn()
        {
                if (sHeader == null)
                        return null;//"ERROR: Please supply credentials";

                string usr = sHeader.Username;
                string pwd = sHeader.Password;

                object [] args = new object[0];

                if( auth.VerifyLogin(usr, pwd,"AFTER_HR_ACCESS"))
                {
                        return DataAccess.FetchEmpInfo(usr,pwd);
                }
                return null;
        }

        [WebMethod]
        public UserInfo EditPersonalDetails(string usr, string pwd)
        {
                if( auth.VerifyAccess("EDIT_ACCT_INFO"))
                {
                        return DataAccess.FetchUserInfo(usr,pwd);
}
                return null;
        }
```

```
        /*Web Service that returns transactions list*/
        [WebMethod]
        public TrxnDS ViewTransactions( string stDt, string endDt )
        {
                DateTime startDate = DateTime.Parse(stDt);
                DateTime endDate = DateTime.Parse(endDt);
                if( auth.VerifyAccess("EDIT_TRXN"))
                {
                        return DataAccess.GetTrxn(startDate, endDate);
                }
                return null;
        }

        // Service to check whether employee has permission to open a
// new customer account
        [WebMethod]
        public bool OpenAcct()
        {
                return auth.VerifyAccess("OPEN_ACT");
        }

        // Service to Open a new customer account
        [WebMethod]
        public int OpenAcctAddDetails(UserInfo acc)
        {
                return DataAccess.InsertUserInfo(acc);
        }

        // Service to deposit money in customer account
        [WebMethod]
        public double Deposit(AcctInfo acc)
        {
                return DataAccess.Deposit(acc);
        }
        [WebMethod]
        public bool EditTrxn()
        {
                if( auth.VerifyAccess("EDIT_TRXN"))
                {
                        return true;
                }
                return false;
        }
```

```
        [WebMethod]
        public Trxn FetchTrxn(int trxnId)
        {
            if( auth.VerifyAccess("EDIT_TRXN"))
            {
                return DataAccess.FetchTrxn(trxnId);
            }
            return null;
        }

        [WebMethod]
        public double UpdateTrxn(Trxn trxnInfo, double oldTrxnAmt)
        {
            return DataAccess.UpdateTrxn(trxnInfo,oldTrxnAmt);

        }

} // end of BOC class
    public class AuthHeaderCS:System.Web.Services.Protocols.SoapHeader
        {
            private string _userName;
            private string _password;

            public AuthHeaderCS() {}
            public string Username
            {
                get { return _userName; }
                set { _userName=value; }
            }
            public string Password
            {
                get { return _password; }
                set { _password=value; }
            }

        }
}
```

********** DataAccess.cs **********

```
using System;
using System.Data;
using System.Configuration;
using Microsoft.ApplicationBlocks.Data;
using System.Data.SqlClient;
```

```
namespace BOC
{
	public class DataAccess
	{
		public static readonly string connStr;
		private static SqlConnection conn;

		public DataAccess()
		{}

		static DataAccess()
		{
			connStr =
ConfigurationSettings.AppSettings["ConnectionString"];
		}

public static TrxnDS GetTrxn( DateTime startDate, DateTime endDate)
		{
			TrxnDS trxnInfo = new TrxnDS();
			conn = new SqlConnection(connStr) ;
			SqlParameter [] dateParams = new SqlParameter[2];

			dateParams[0] = new
SqlParameter("@startDate",SqlDbType.DateTime);
			dateParams[0].Value = startDate;

			dateParams[1] = new
SqlParameter("@endDate",SqlDbType.DateTime);
			dateParams[1].Value = endDate;

		try	{
		SqlHelper.FillDataset( conn,CommandType.StoredProcedure,
			"GetTransactionList", trxnInfo, new
			string[]{"Transaction"}, dateParams);

			}
			catch
			{
				throw;
			}
			finally
			{
				if(conn != null)
```

```
                    conn.Dispose();
                }
                return trxnInfo;
            }

            public static int InsertUserInfo( UserInfo acc )
            {
                int acctNo=0;
                int usrId = 0;
                SqlParameter[] accountParms ;    //=
                GetAccountParameters();

            accountParms  = new SqlParameter[] {
                new SqlParameter("@UserName", SqlDbType.VarChar, 20),
                new SqlParameter("@FirstName", SqlDbType.VarChar, 20),
                new SqlParameter("@LastName", SqlDbType.VarChar, 20),

                new SqlParameter("@Email", SqlDbType.VarChar, 20),
                new SqlParameter("@Password", SqlDbType.VarChar, 20),
                new SqlParameter("@DOB", SqlDbType.DateTime),
                new SqlParameter("@RoleId", SqlDbType.Int),
                new SqlParameter("@Adrr1", SqlDbType.VarChar, 25),
                new SqlParameter("@Adr2", SqlDbType.VarChar, 25),
                new SqlParameter("@City", SqlDbType.VarChar, 25),

                new SqlParameter("@State", SqlDbType.VarChar, 2),
                new SqlParameter("@Zip", SqlDbType.VarChar, 5),
                new SqlParameter("@phone", SqlDbType.VarChar, 20),
                new SqlParameter("@UserId", SqlDbType.Int)
                };
            accountParms[13].Direction = ParameterDirection.Output;
            SetAccountParameters(accountParms, acc);
            using (SqlConnection conn = new SqlConnection(connStr))
            {
              conn.Open();
              try
        {
                    SqlHelper.ExecuteNonQuery(conn,
CommandType.StoredProcedure, "OpenAcct", accountParms);
                    usrId = int.Parse(accountParms[13].Value.ToString());
              }
             catch
                    {
                        throw;
```

```
                    }
                }
                    }

        SqlParameter [] loginParams =  new SqlParameter[]
        {
                new SqlParameter("@UserId",SqlDbType.Int),
                new SqlParameter("@ActType",SqlDbType.Char),
                new SqlParameter("@Balance",SqlDbType.Float),
                new SqlParameter("@AccountNo",SqlDbType.Int)
        };
        loginParams[0].Value = usrId;
        loginParams[1].Value = acc.AccountInfo.ActType;
        loginParams[2].Value = acc.AccountInfo.Balance;
        loginParams[3].Direction = ParameterDirection.Output;
        using (SqlConnection conn = new SqlConnection(connStr))
        {
        conn.Open();

        try
        {
                SqlHelper.ExecuteNonQuery(conn, CommandType.StoredProcedure,
                "InsertAccountInfo", loginParams);
                acctNo = int.Parse(loginParams[3].Value.ToString());
        }
        catch
        {
                //trans.Rollback();
                throw;
         }
        }
   return acctNo;
}
public static int Deposit( AcctInfo acc )
{
        double balance = GetBalanceAmt( acc.ActNo, acc.ActType );
        double newBalance = balance + acc.Balance;
        int trxnId=0;

        SqlParameter[] acctParms = new SqlParameter[3];

        acctParms[0] = new SqlParameter("@AccountNo", SqlDbType.Int);
        acctParms[0].Value = acc.ActNo;
        acctParms[0].Direction = ParameterDirection.Input;
```

```
acctParms[1] = new SqlParameter("@actType", SqlDbType.Char);
acctParms[1].Value = acc.ActType;
acctParms[1].Direction = ParameterDirection.Input;

acctParms[2] = new SqlParameter("@Balance", SqlDbType.Float);
                        acctParms[2].Value = newBalance;

SqlParameter[] trxnParms = new SqlParameter[7];
trxnParms[0] = new SqlParameter("@SrcAcctNo", SqlDbType.Int);
trxnParms[0].Value = acc.ActNo;
trxnParms[1] = new SqlParameter("@DestAcctNo", SqlDbType.Int);
trxnParms[1].Value = acc.ActNo;
trxnParms[2] = new SqlParameter("@TrxType", SqlDbType.NVarChar);
trxnParms[2].Value = "D";
trxnParms[3] = new SqlParameter("@Amount", SqlDbType.Float);
trxnParms[3].Value = acc.Balance;
trxnParms[4] = new SqlParameter("@TrxDate", SqlDbType.DateTime);
trxnParms[4].Value = DateTime.Now;
trxnParms[5] = new SqlParameter("@Notes", SqlDbType.NVarChar);
trxnParms[5].Value = "Deposit";
trxnParms[6] = new SqlParameter("@TrxId", SqlDbType.Int);
trxnParms[6].Value = 0;
trxnParms[6].Direction = ParameterDirection.Output;

using (SqlConnection conn = new SqlConnection(connStr))
{
        conn.Open();
        using (SqlTransaction trans = conn.BeginTransaction())
        {
        try
        {
                SqlHelper.ExecuteNonQuery(trans,
                CommandType.StoredProcedure, "UpdateAccountBalance",
                acctParms);
                SqlHelper.ExecuteNonQuery(trans,
                CommandType.StoredProcedure, "InsertTransaction", trxnParms);
                trxnId= int.Parse(trxnParms[6].Value.ToString());
                trans.Commit();
        }
        catch
        {
                trans.Rollback();
                newBalance=0;
```

```
                }
            }
        }
                return trxnId;
}


public static double GetBalanceAmt( int actNo, string actType )
{
        SqlDataReader rdr = null;
        conn = new SqlConnection(connStr) ;
        double balance = 0;


        SqlParameter[] acctParms = new SqlParameter[2];

        acctParms[0] = new SqlParameter("@AccountNo", SqlDbType.Int);
        acctParms[0].Value = actNo;

        acctParms[1] = new SqlParameter("@actType", SqlDbType.Char);
                        acctParms[1].Value = actType;
        try
        {
                try
                {
                        conn.Open();
                }
                catch
                {
                        conn.Close();
                        throw;

                }
                rdr =   SqlHelper.ExecuteReader(conn,
                CommandType.StoredProcedure,"GetBalance", acctParms);

                if (rdr.Read())
                {
                        balance = rdr.GetDouble(0);
                }

    }
        catch
        {
```

```
                throw;
        }
        finally
        {
                if(rdr != null)
                        ((IDisposable)rdr).Dispose();

                if(conn != null)
                        conn.Dispose();
        }
  return balance;
}

private static void SetAccountParameters(SqlParameter[] parms, UserInfo acc)
{
        parms[0].Value = acc.UserName;
        parms[1].Value = acc.Address.FirstName;
        parms[2].Value = acc.Address.LastName ;
        parms[3].Value = acc.Email ;
        parms[4].Value = acc.Password;
        parms[5].Value = acc.DOB; //"1/1/1990";
        parms[6].Value = acc.Role;
        parms[7].Value = acc.Address.Address1;
        parms[8].Value = acc.Address.Address2;
        parms[9].Value = acc.Address.City;
        parms[10].Value = acc.Address.State;
        parms[11].Value = acc.Address.Zip;
        parms[12].Value = " ";
        parms[13].Value = acc.UserId; //Convert.ToInt32(0);
}

public static EmployeeInfo FetchEmpInfo( string username, string password)
{
        SqlDataReader rdr = null;
        conn = new SqlConnection(connStr) ;

        DataSet ds = new DataSet();
        SqlParameter [] roleParams = new SqlParameter[1];
        roleParams[0] = new SqlParameter("@roleId", SqlDbType.Int);
        SqlParameter [] loginParams = new SqlParameter[2];

        loginParams[0] = new SqlParameter("@username",SqlDbType.NVarChar, 20);
        loginParams[0].Value = username;
        loginParams[1] = new SqlParameter("@password", SqlDbType.NVarChar, 10);
```

```
        loginParams[1].Value = password;
    try
                {
                try
                {
                conn.Open();
                }
                catch
                {
                conn.Close();
                throw;

                }
                        rdr =   SqlHelper.ExecuteReader(conn,
                        CommandType.StoredProcedure,"GetEmployee", loginParams);

                if (rdr.Read())
                {
                 return new EmployeeInfo(rdr.GetInt32(0), rdr.GetString(1),
                                                rdr.GetString(2),rdr.GetString(3),
                                                rdr.GetString(4)  );
                }
                }
                catch
                {
                        throw;
                }
                finally
                {
                        if(rdr != null)
                                ((IDisposable)rdr).Dispose();

                        if(conn != null)
                                conn.Dispose();
                }
     return null;
}


        // Fetch user details to verify role and access
        public static UserInfo FetchUserInfo( string username, string password)
        {
                SqlDataReader rdr = null;
                conn = new SqlConnection(connStr) ;
```

```
        DataSet ds = new DataSet();
        SqlParameter [] roleParams = new SqlParameter[1];
        roleParams[0] = new SqlParameter("@roleId", SqlDbType.Int);
        SqlParameter [] loginParams = new SqlParameter[2];

        loginParams[0] = new SqlParameter("@username",SqlDbType.NVarChar,
        20);
        loginParams[0].Value = username;
        loginParams[1] = new SqlParameter("@password",
        SqlDbType.NVarChar, 10);
        loginParams[1].Value = password;

        try
        {
        try
        {
                conn.Open();
        }
        catch
        {
                conn.Close();
                throw;

        }
        rdr =  SqlHelper.ExecuteReader(conn,
        CommandType.StoredProcedure,"GetUserInfo", loginParams);

if (rdr.Read())
{
        AddressInfo addr = new AddressInfo(rdr.GetString(1),
                        rdr.GetString(2), rdr.GetString(4),
                        rdr.GetString(5), rdr.GetString(6),
                        rdr.GetString(7), rdr.GetString(8));

    AcctInfo actInf = new AcctInfo();

        return new UserInfo(rdr.GetInt32(0),username, password,
                        rdr.GetString(3),rdr.GetDateTime(10),
                        addr, actInf,  rdr.GetInt32(9)  );

  }
}
catch
```

```
        {
                throw;
        }
        finally
        {
                if(rdr != null)
                        ((IDisposable)rdr).Dispose();
                if(conn != null)
                        conn.Dispose();
        }
        return null;

}

// Fetch trxn  details for the given trxnId
public static Trxn FetchTrxn( int trxnId)
{
        SqlDataReader rdr = null;
        conn = new SqlConnection(connStr) ;

        SqlParameter [] loginParams = new SqlParameter[1];

        loginParams[0] = new SqlParameter("@TrxId",SqlDbType.Int);
        loginParams[0].Value = trxnId;


        try
        {
         try
         {
                conn.Open();
         }
         catch
         {
                conn.Close();
                throw;
         }
         rdr =  SqlHelper.ExecuteReader(conn,
         CommandType.StoredProcedure,"GetTransaction", loginParams);

         if (rdr.Read())
         {

                return new Trxn(trxnId, rdr.GetInt32(0), rdr.GetInt32(1),
```

```
                    rdr.GetString(2),rdr.GetDouble(3),
                    rdr.GetDateTime(4), rdr.GetString(5));
        }

        }
        catch
        {
                throw;
        }
        finally
        {
                if(rdr != null)
                ((IDisposable)rdr).Dispose();

        if(conn != null)
                conn.Dispose();
        }
        return null;
}


public static double UpdateTrxn( Trxn trxn, double oldTrxnAmt)
{
        double balance = GetBalanceAmt( trxn.SrcActNo, "C" );
        double origAmt = balance - oldTrxnAmt;
        double newBalance = origAmt + trxn.Amount;


        SqlParameter[] trxnParms = new SqlParameter[7];
        trxnParms[0] = new SqlParameter("@TrxId", SqlDbType.Int);
        trxnParms[0].Value = trxn.TrxnId;
        trxnParms[1] = new SqlParameter("@SrcAcctNo", SqlDbType.Int);
        trxnParms[1].Value = trxn.SrcActNo;
        trxnParms[2] = new SqlParameter("@DestAcctNo", SqlDbType.Int);
        trxnParms[2].Value = trxn.DestActNo;
        trxnParms[3] = new SqlParameter("@TrxType", SqlDbType.NVarChar);
        trxnParms[3].Value = trxn.TrxnType;
        trxnParms[4] = new SqlParameter("@Amount", SqlDbType.Float);
        trxnParms[4].Value = trxn.Amount;
        trxnParms[5] = new SqlParameter("@TrxDate", SqlDbType.DateTime);
        trxnParms[5].Value = DateTime.Now;
        trxnParms[6] = new SqlParameter("@Notes", SqlDbType.NVarChar);
        trxnParms[6].Value = trxn.Notes;
```

```
    SqlParameter[] acctParms = new SqlParameter[3];

    acctParms[0] = new SqlParameter("@AccountNo", SqlDbType.Int);
    acctParms[0].Value = trxn.SrcActNo;


    acctParms[1] = new SqlParameter("@ActType", SqlDbType.Char);
    acctParms[1].Value = "C";


    acctParms[2] = new SqlParameter("@Balance", SqlDbType.Float);
    acctParms[2].Value = newBalance;

    if ( newBalance > 0 )
    {
    using (SqlConnection conn = new SqlConnection(connStr))
    {
      conn.Open();
      using (SqlTransaction trans = conn.BeginTransaction())
      {
  try
       {
            SqlHelper.ExecuteNonQuery(trans, CommandType.StoredProcedure,
            "UpdateTrxnAmt", acctParms);
            SqlHelper.ExecuteNonQuery(trans, CommandType.StoredProcedure,
            "UpdateTransaction", trxnParms);

          trans.Commit();
      }
          catch
          {
                trans.Rollback();
                return 0;
          }
     }
    }
   }                else                    {
   newBalance = 0.00;
 }
 return newBalance;


}
```

```
  }// end of DataAccess class
}


********** AddressInfo.cs **********
using System;

namespace BOC
{
        /// <summary>
        /// Summary description for AddressInfo.
        /// </summary>
        public class AddressInfo
        {

                // Internal member variables
                private string _firstName;
                private string _lastName;
                private string _address1;
                private string _address2;
                private string _city;
                private string _state;
                private string _zip;



                /// <summary>
                /// Default constructor
                /// </summary>
                public AddressInfo()
                {
                }

                /// <summary>
                /// Constructor with specified initial values
                /// </summary>
                /// <param name="firstName">First Name</param>
                /// <param name="lastName">Last Name</param>
                /// <param name="address1">Address line 1</param>
                /// <param name="address2">Address line 2</param>
                /// <param name="city">City</param>
                /// <param name="state">State</param>
                /// <param name="zip">Postal Code</param>
                /// <param name="country">Country</param>
```

```
        /// <param name="phone">Phone number at this address</param>
        public AddressInfo(string firstName, string lastName, string address1,
string address2, string city, string state, string zip)
        {
            this._firstName = firstName;
            this._lastName = lastName;
            this._address1 = address1;
            this._address2 = address2;
            this._city = city;
            this._state = state;
            this._zip = zip;

        }

        // Properties
        public string FirstName
        {
            get { return _firstName; }
            set { _firstName = value; }
        }

        public string LastName
        {
            get { return _lastName; }
            set { _lastName = value; }
        }

        public string Address1
        {
            get { return _address1; }
            set { _address1 = value; }
        }

        public string Address2
        {
            get { return _address2; }
            set { _address2 = value; }
        }

        public string City
        {
            get { return _city; }
            set { _city = value; }
        }
```

```
                public string State
                {
                        get { return _state; }
                        set { _state = value; }
                }

                public string Zip
                {
                        get { return _zip; }
                        set { _zip = value; }
                }


        }
}
```

********** AcctInfo.cs **********

```
using System;

namespace BOC
{
        /// <summary>
        /// Summary description for AcctInfo.
        /// </summary>
        public class AcctInfo
        {
                // Internal member variables
                private int _actNo;
                private double _balance;
                private string _actType;
                private int _userId;



                public AcctInfo()
                {
                }

                public AcctInfo(int actNo, float balance,string actType, int userId)
                {
                        this._actNo=actNo;
                        this._balance=balance;
                        this._actType=actType;
```

```
                this._userId=userId;
        }

        public int ActNo
        {
                get { return _actNo; }
                set { _actNo = value; }
        }

        public int UserId
        {
                get { return _userId; }
                set { _userId = value; }
        }

        public string ActType
        {
                get { return _actType; }
                set { _actType = value; }
        }

        public double Balance
        {
                get { return _balance; }
                set { _balance = value; }
        }
    }
}
```

**********Transaction.cs **********

```
using System;

namespace BOC
{
    /// <summary>
    /// Summary description for Trxn.
    /// </summary>
    public class Trxn
    {
        private int _trxnId;
        private int _srcActNo;
        private int _destActNo;
        private string  _trxnType;
        private double _amount;
        private DateTime  _date;
```

```
        private string _notes;

        public Trxn()
        {
        }

        public Trxn( int trxnId, int srcActNo, int destActNo, string trxnType,
double amount, DateTime date, string notes  )
        {
                this._trxnId = trxnId;
                this._srcActNo = srcActNo;
                this._destActNo = destActNo;
                this._trxnType = trxnType;
                this._amount = amount;
                this._date = date;
                this._notes = notes;
        }

        public int TrxnId
        {
                get { return _trxnId; }
                set { _trxnId = value; }
        }
        public int SrcActNo
        {
                get { return _srcActNo; }
                set { _srcActNo = value; }
        }

        public int DestActNo
        {
                get { return _destActNo; }
                set { _destActNo = value; }
        }
        public string TrxnType
        {
                get { return _trxnType; }
                set { _trxnType = value; }
        }

        public double Amount
        {
                get { return _amount; }
                set { _amount = value; }
```

```
        }
        public DateTime TrxnDate
        {
            get { return _date; }
            set { _date = value; }
        }

        public string Notes
        {
            get { return _notes; }
            set { _notes = value; }
        }
    }
}
```